AN OBJECT ORIENTED VIDEO SYSTEM

Publication number: WO0131497

Publication date: 2001-05-03

Inventor: GONZALEZ RUBEN (AU)

Applicant: ACTIVESKY INC (US); GONZALEZ RUBEN (AU)

Classification:

- international: *G06F17/30; H04L12/28; H04L12/56; H04N7/26; H04N7/28; H04N7/52; H04N7/66; H04L29/06; H04N7/16;* G06F17/30; H04L12/28; H04L12/56; H04N7/26; H04N7/52; H04N7/64; H04L29/06; H04N7/16; (IPC1-7): G06F17/30; H04L12/56; H04N7/26

- european: H04N7/26J6; G06F17/30E; G06F17/30S3; H04L12/28W; H04L12/56B; H04N7/24C12C; H04N7/26A6C8; H04N7/26J8; H04N7/26J12; H04N7/26Z10; H04N7/28; H04N7/52; H04N7/66

Application number: WO2000AU01296 20001020

Priority number(s): AU1999PQ03603 19991022; AU2000PQ08661 20000707

Also published as:

- EP1228453 (A1)
- MXPA02004015 (A)
- EP1228453 (A0)
- CA2388095 (A1)

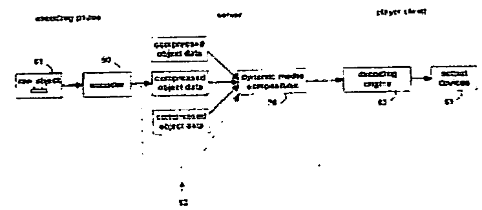
Cited documents:

- US5862325
- US5586235
- WO9736376
- EP0240948
- WO0026857

more >>

Abstract of **WO0131497**

A method of generating an object oriented interactive multimedia file, including encoding data comprising at least one of video, text, audio, music and/or graphics elements as a video packet stream, text packet stream, audio packet stream, music packet stream and/or grahics packet stream respectively, combining the packet streams into a single self-contained object, said object containing its own control information, placing a plurality of the objects in a data stream, and grouping one or more of the data streams in a single contiguous self-contained scene, the scene including format definition as the initial packet in a sequence of packets. An encoder for executing the method is provided together with a player or decoder for parsing and decoding the file, which can be wirelessly streamed to a portable computer device, such as a mobile phone or a PDA. The object controls provide rendering and interactive controls for objects allowing users to control dynamic media composition, such as dictating the shape and content of interleaved video objects, and control the objects received.



AN OBJECT ORIENTED VIDEO SYSTEM

Description of WO0131497

AN OBJECT ORIENTED VIDEO SYSTEMField of the Invention
The present invention relates to a video encoding and processing method, and in particular, but not exclusively, to a video encoding system which supports the coexistence of multiple arbitrarily-shaped video objects in a video scene and permits individual animations and interactive behaviours to be defined for each object, and permits dynamic media composition by encoding object oriented controls into video streams that can be decoded by remote client or standalone systems. The client systems may be executed on a standard computer or on mobile computer devices, such as personal digital assistants (PDAs), smart wireless phones, hand-held computers and wearable computing devices using low power, general purposeCPUs. These devices may include support for wireless transmission of the encoded video streams.

Background
Recent technology improvements have resulted in the introduction of personal mobile computing devices, which are just beginning to include full wireless communication technologies. The global uptake of wireless mobile telephones has been significant, but still has substantial growth potential. It has been recognised that there have not been any video technology solutions that have provided the video quality, frame rate or low power consumption for potential new and innovative mobile video processes. Due to the limited processing power of mobile devices, there are currently no suitable mobile video solutions for processes utilising personal computing devices such as mobile video conferencing, ultra-thin wireless network client computing, broadcast wireless mobile video, mobile video promotions or wireless video surveillance.

A serious problem with attempting to display video on portable handheld devices such as smart phones and PDAs is that in general these have limited display capabilities. Since video is generally encoded as using continuous colour representation which requires true colour (16 or 24 bit) display capabilities for rendering, severe performance degradation results when an 8 bit display is used. This is due to the quantisation and dithering processes that are performed on the client to convert the video images into an 8 bit format suitable for display on devices using a fixed colour map, which reduces quality and introduces a large processing overhead.

Computer based video conferencing currently uses standard computer workstations or PCs connected through a network including a physical cable connection and network computer communication protocol layers. An example of this is a videoconference between two PCs over the Internet, with physically connected cables end to end, using the TCP/IP network communication protocols. This kind of video conferencing has a physical connection to the Internet, and also uses large, computer-based video monitoring equipment. It provides for a videoconference between fixed locations, which additionally constrains the participants to a specific time for the conference to ensure that both parties will be at the appropriate locations simultaneously.

Broadcast of wireless textual information for personal handheld computers or smartphones has only recently become feasible with advances in new and innovative wireless technologies and handheld computing devices. Handheld computing devices and mobile telephones are able to have wireless connections to wide area networks that can provide textual information to the user device. There is currently no real-time transmission of video to wireless handheld computing devices. This lack of video content connectivity tends to limit the commercial usefulness of existing systems, especially when one considers the inability of"broadcast"systems to target specific users for advertising purposes. One important market issue for broadcast media in any form is the question of advertising and how it is to be supported. Effective advertising should be specifically targeted to users and geographic locations, but broadcast technologies are inherently limited in this regard. As a consequence,"niche"advertisers of specialty products would be reluctant to support such systems.

Current video broadcast systems are unable to embed targeted advertising because of the considerable processing requirements needed to insert advertising material into video data streams in real time during transmission. The alternate method of pre-compositing video prior to transmission is too tedious as recognised by the present inventor to be performed on a regular basis. Additionally, once the advertising is embedded into the video stream, the user is unable to interact with the advertising which, reduces the effectiveness of the advertising. Significantly, it has been recognised that more effective advertising can be achieved though interactive techniques.

Most video encoders/decoders exhibit poor performance with cartoons or animated content; however, there is more cartoon and animated content being produced for the
Internet than video. It has been recognised that there is a need for a codec which enables efficient encoding of graphics animations and cartoons as well as video.

Commercial and domestic security-based video surveillance systems have to date been achieved using closed circuit monitoring systems with video monitoring achieved in a central location, requiring the full-time attention of a dedicated surveillance guard. Video monitoring of multiple locations can only be achieved at the central control centre using dedicated monitoring system equipment. Security guards have no access to video from monitored locations whilst on patrol.

Network-based computing using thin client workstations involves minimal software processing on the client workstation, with the majority of software processing occurring on a server computer. Thin client computing reduces the cost of computer management due to the centralisation of information and operating software configuration. Client workstations are physically wired through standard local area networks such as 10 Base T
Ethernet to the server computer. Client workstations run a minimal operating system, enabling communication to a backend server computer and information display on the client video monitoring equipment. Existing systems, however, are constrained. They are typically limited to specific applications or vendor software. For example, current thin clients are unable to simultaneously service a video being displayed and a spreadsheet application.

In order to directly promote product in the market, sales representatives can use video demonstrations to illustrate product usage and benefits. Currently, for the mobile sales representative, this involves the use ofcumbersome dedicated video display equipment, which can be taken to customer locations for product demonstrations. There are no mobile handheld video display solutions available, which provide real-time video for product and market promotional purposes.

Video brochures have often been used for marketing and advertising. However, their effectiveness has always been limited because video is classically a passive medium. It has been recognised that the effectiveness of video brochures would be dramatically improved if they could be made interactive. If this interactivity could be provided intrinsically within a codec, this would open the door to video-based e-commerce applications. The conventional definition for interactive video includes a player that is able to decompress a normal compressed video into a viewing window and interpret some metadata which defines buttons and invisible"hot regions"to be overlaid over the video, typically representing hyperlinks where a user's mouse click will invoke some predefined action. In this typical approach, the video is stored as a separate entity from the metadata, and the nature of interaction is extremely limited, since there is no integration between the video content and the external controls that are applied.

The alternative approach for providing interactive video is that of MPEG4, which permits multiple objects, however this approach finds difficulty running on todays typical desktop computer such as a Pentium III 500 Mhz Computer having 128 Mb RAM. The reason being that the object shape information is encoded separately from the object colour/luminance information generating additional storage overhead, and that the nature of the scene description (BIFS) and file format having been taken in part from virtual reality markup language (VRML) is very complex. This means that to display each video frame for a video object three separate components have to be fully decoded ; the luminance information, the shape/transparency information and the BIFS. These then have to be blended together before the object can be displayed. Given that the DCT based video codec itself is already very computationally intensive, the additional decoding requirements introduce significant processing overheads in addition to the storage overheads.

The provision of wireless access compatibilities to personal digital assistants(PDAs) permits electronic books to be freed from their storage limitations by enabling real-time wireless streaming of audio-visual content to PDAs. Many corporate training applications need audiovisual information to be available wirelessly in portable devices. The nature of audiovisual training materials dictates that they be interactive and provide for non-linear navigation of large amounts of stored content. This cannot be provided with the current state of the art.

Objects of the invention
An object of the invention is to overcome the deficiencies described above. Another object of the invention is to provide software playback of streaming video, and to display video on a low processingpower, mobile device such as a general-purpose handheld devices using a general purpose processor, without the aid of specialised DSP or custom hardware.

A further object of the invention is to to provide a high performance low complexity software video codec for wirelessly connected mobile devices. The wireless connection may be provided in the form of a radio network operating in CDMA, TDMA, FDMA transmission modes over packet swithced or circuit switched networks as used in GSM,
CDMA, GPRS, PHS, UMTS, IEEE 802.11 etc networks.

A further object of the invention is to send colour prequantisation data for real-time colour quantisation on clients with 8 bit colour displays (mapping any non-stationary threedimensional data onto a single dimension) when using codecs. that use continuous colour representations.

A further object of the invention is to support multiple arbitrary shaped video objects in a single scene with no extra data overhead or processing overhead.

A further object of the invention is to integrate audio, video, text, music and animated graphics seamlessly into a video scene.

A further object of the invention is to attach control information directly to objects in a video bitstream to define interactive behavior, rendering, composition, digital rights management information, and interpretation of compressed data for objects in a scene.

A further object of the invention is to interact with individual objects in the video and control rendering, and the composition of the content being displayed.

Yet another object of the invention is to provide interactive video possessing the capability of modifying the rendering parameters of individual video objects, executing specific actions assigned to video objects when conditions become true, and the ability to modify the overall system status and perform non-linear video navigation. This is achieved through the control information that is attached to individual objects.

Another object of the invention is to provide interactive non-linear video and composite media where the system is capable of responding in one instance to direct user interaction with hyperlinked objects by jumping to the specified atget scene. In another instance the path taken through given portions of the video is indirectly determined by user interaction with other not directly related objects. For example the system may track what scenes have been viewed previously and automatically determine the next scene to be displayed based on this history.

Interactive tracking data can be provided to the server during content serving. For downloaded content, the interactive tracking data can be stored on the device for later synchronization back to the server. Hyperlink requests or additional information requests selected during replay of content off-line will be stored and sent to the server for fulfillment on next synchronization (asynchronous uploading of forms and interaction data).

A further object of the invention is to provide the same interactive control over object oriented video whether the video data is being streamed from a remote server or being played offline from local storage. This allows the application of interactive video in the following distribution alternatives; streaming ("pull"), scheduled ("push"), and download.

It provides for automatically and asynchronous uploading of forms and interaction data from a client device when using download or scheduled distribution model,
An object of the invention to animate the rendering parameters of audio/visual objects within a scene. This includes, position, scale, orientation, depth, transparency, colour, and volume. The invention aims to achieve this through defining fixed animation paths for rendering parameters, sending commands from a remote server to modify the rendering parameters, and changing the rendering parameters as a direct or indirect consequence of user interaction, such as activating an animation path when a user clicks on an object.

Another object of the invention is to define behaviours to individual audio-visual objects that are executed when users interact with objects, wherein the behaviours include animations, hyper-linking, setting of system states/variables, and control of dynamic media composition.

Another object of the invention is to conditionally execute immediate animations or behavioural actions on objects. These conditions may include the state of system variables, timer events, user events and relationships between objects (e. g., overlapping), the ability to delay these actions until conditions become true, and the ability to define complex conditional expressions. It is further possible to retarget any control from one object to another so that interaction with one object affects another rather than itself.

Another object of the invention includes the ability to create video menus and simple forms for registering user selections. Said forms being able to be automatically uploaded to a remote server synchronously if online or asynchronously if the system off-line.

An object of the invention is to provide interactive video, which includes the ability to define loops; such as looping the play of an individual object's content or looping of object control information or looping entire scenes.

Another object of the invention is to provide multi-channel control where subscribers can change the viewed content stream to another channel such as to/from a unicast (packet switched connection) session from/to a multicast (packet or circuit switched) channel. For example interactive object behaviour may be used to implement a channel changing feature where interacting with an object executes changing channels by changing from a packet switched to circuit switched connections in devices supporting both connection modes and changing between unicast and broadcast channels in a circuit switched connection and back again.

Another object of the invention is to provide content personalisation through dynamic media composition ("DMC") which is the process of permitting the actual content of a displayed video scene to be changed dynamically, in real-time while the scene is being viewed, by inserting, removing or replacing any of the arbitrary shaped visual/audio video objects that the scene includes, or by changing the scene in the video clip.

An example would be an entertainment video containing video object components, which relate to the subscribers user profile. For example in a movie scene, a room could contain golf sporting equipment rather than tennis. This would be particularly useful in advertising media where there is a consistent message but with various alternative video object components.

Another object of the invention is to enable the delivery and insertion of a targeted inpicture interactive advertising video object with or without interactive behaviour into a viewed scene as an embodiment of the dynamic media process.. The advertising object may be targeted to the user based on time of day, geographic location, user profile etc.

Furthermore, the invention aims to allow for the handling of various kinds of immediate or delayed interactive response to user interaction (eg a user click) with said object including removal of advertisement, performing a DMC operation such as immediately replacing the advertising object with another object or replacing the viewed scene with a new one, registering the user for offline follow-up actions, and jumping to a new hyperlink destination or connection at the end of the current video scene/session, or and changing the transparency of the advertising object or making it go away or disappear. Tracking of user interaction with advertisment objects when these are provided in a real-time streaming scenario further permits customisation of targetting purposes or evaluation of advertising effectiveness.

Another object of the invention is to subsidise call charges associated with wireless network or smartphone use through advertising by automatically displaying a sponsor's video advertising object for a sponsored call during or at the end of a call. Alternatively, displaying an interactive ivdeo object prior to, during or after the call offering sponsorship if the user performs some interaction with the object.

An object of the invention is to provide a wireless interactive e-commerce system for mobile devices using audio and visual data in online and off-line scenarios. The ecommerce include marketing/promotional purposes using either hyper-linked in-picture advertising or interactive video brochures with nonliner navigation, or direct online shopping where individual sale items can be created as objects so that users may interact with them such as dragging them into shopping baskets etc.

An object of the invention includes a method and system to freely provide to the public, (or at subsidised cost), memory devices such as compact flash or memory stick or a memory devices having some other form factor that contains interactive video brochures with advertising or promotional material or product information. The memory devices are preferably read only devices, although other types of memory can be used. The memory devices may be configured to provide a feedback mechanism to the producer, using either online communication, or by writing some data back on to the memory card which is then deposited at some collection point. Without using physical memory cards, this same objective may be accomplised using local wireless distribution by pushing information to devices following negotiation with the device regarding if the device is prepared to receive the data and the quantity receivable.

An object of the invention is to send to users when in download, interactive video brochures,videozines and video (activity) books so that they can then interact with the brochures including filling out forms, etc. If present in the video brochure and actioned or interacted by a user, userdata/forms these will then be asynchronously uploaded to the originating server when the client becomes online again. If desired, the uploading can be performed automatically and/or asynchronously. These brochures may contain video for training/educational, marketing or promotional, product information purposes and the collected. user interaction information may be a test, survey, request for more information, purchase order etc. The interactive video brochures, videozines and video (activity) books may be created with in-picture advertising objects.

A further object of the invention is to create unique video based user interfaces for mobile devices using our object based interactive video scheme.

A further object of the invention is to provide video mail for wirelessly connected mobile users where electronic greeting cards and messages may be created and customised and forwarded among subscribers.

A further object of the invention is to provide local broadcast as in sports arenas or other local environments such as airports, shopping malls with back channel interactive user requests for additional information or e-commerce transactions.

Another object of the invention is to provide a method for voice command and control of online applications using the interactive video systems.

Another object of the invention is to provide a wireless ultrathin clients to provide access to remote computing servers via wireless connections. The remote computing server may be a privately owned computer or provided by an application service provider.

Still another object of the invention is to provide videoconferencing including multiparty video conferencing on low-end wireless devices with or without in-picture advertising.

Another object of the invention is to provide a method of video surveillance, whereby a wireless video surveillance system inputs signals from video cameras, video storage devices, cable TV and broadcast TV, streaming internet video for remote viewing on a wirelessly connected PDA or mobile phone. Another object of the invention is to provide a traffic monitoring service using a street traffic camera.

Summary of the Invention

System/Codec Aspects

The invention provides the ability to stream and/or run video on low-power mobile devices in software, if desired. The invention further provides the use of a quadtree-based codec for colour mapped video data. The invention further provides using a quadtreebased codec with transparent leaf representation, leaf colour prediction using a FIFO, bottom level node type elimination, along with support for arbitrary shape definition.

The invention further includes the use of a quadtree based codec with nth order interpolation for non-bottom leaves and zeroth order interpolation on the bottom level leaves and support for arbitrary shape definition. Thus, features of various embodiments of the invention may include one or more of the following features: sending colour prequantisation information to permit real-time client side colour quantisation; using a dynamic octree datastructure to represent the mapping of a 3D data spacing into an adaptive codebook for vector quantisation; the ability to seamlessly integrating audio, video, text, music and animated graphics into a wireless streaming video scene; supporting multiple arbitrary shaped video objects in a single scene. This feature is implemented with no extra data overhead or processing overhead, for example by encoding additional shape information separate from luminance or texture information; basic file format constructs, such as file entity hierarchy, object data streams, separate specification of rendering, definition and content parameters, directories, scenes, and object based controls; the ability to interact with individual objects in wireless streaming video; the ability to attach object control data to objects in the video bit streams to control interaction behaviour, rendering parameters, composition etc; the ability to embed digital rights management information into video or graphic animation data stream for wireless streaming based distribution and for download and play based distribution; the ability to creating video object user interfaces("VUI's") instead of conventional graphic user interfaces (GUI's); and/or the ability to use an XML based markup language ("IAVML") or similar scripts to define object controls such as rendering parameters and programmatic control of DMC functions in multimedia presentations.

Interaction Aspects

The invention further provides a method and system for controlling user interaction and animation (self action) by supporting

-a method and system for sending object controls from a streaming server to modify data content or rendering of content.

-embedding object controls in a data file to modify data content or rendering of content.

-the client may optionally execute actions defined by the object controls based on direct or indirect user interaction.

The invention further provides the ability to attach executable behaviours to objects, including: animation of rendering parameters, for audio/visual objects in video scenes. hyperlinks, starting timers, making voice calls, dymaic media composition actions, changing system states (e. g., pause/play), changing user variables (e. g., setting a boolean flag).

The invention also provides the ability to activate object behaviours when users specifically interact with objects (e. g., click on an object or drag anobject) when user events occur (paused button pressed, or key pressed), or when system events occur (e. g., end of scene reached).

The invention further provides a method and system for assigning conditions to actions and behaviours these conditions include timer events (e. g., timer has expired), user events (e. g., key pressed), system events (e. g., scene 2 playing), interaction events (e. g., user clicked on object), relationships between objects (e. g., overlapping), user variables (e. g., boolean flag set), and system status (e. g., playing or paused, streaming or standalone play).

Moreover, the invention provides the ability to form complex conditional expressions using AND-OR plane logic, waiting for conditions to become true before execution of actions, the ability to clear waiting actions, the ability to retarget consequences of interactions with objects and other controls from one object to another, permit objects to be replaced by other objects while playing based on user interaction, and/or permit the creation or instantiation of new objects by interacting with an existing object.

The invention provides the ability to define looping play of object data (i. e., frame sequence for individual objects), object controls (i. e., rendering parameters), and entire scenes (restart frame sequences for all objects and controls).

Further, the invention provides the ability to create forms for user feedback or menus for user control and interaction in streaming mobile video and the ability to drag video objects on top of other objects to effect system state changes.

Dynamic Media Composition
The invention provides the ability to permit the composition of entire videos by modifying scenes and the composition of entire scenes by modifying objects. This can be performed in the case of online streaming, playing video off-line (stand-alone), and hybrid.

Individual in-picture objects may be replaced by another object, added to the current scene, and deleted from the current scene.

DMC can be performed in the three modes including fixed, adaptive, and user mediated.

A local object library for DMC support can be used to store objects for use in DMC, store objects for direct playing, that can be managed from a streaming server (insert, update, purge), and that can be queried by the server. Additionally the a local object library for DMC support has versioning control for library objects, automatic expiration of non persistent library objects, and automatic object updating from the server. Furthermore, the invention includes multilevel access control for library objects, supports a unique ID for each library object, has a history or status of each library object, and can enable the sharing of specific media objects between two users.

Further Applications
The invention provides ultrathin clients that provide access to remote computing servers via wireless connections, permit users to create, customise and send electronic greeting cards to mobile smart phones, the use of processing spoken voice commands to control the video display, the use of interactive streaming wireless video from a server for training/educational purposes using non-linear navigation, streaming cartoons/graphic animation to wireless devices, wireless streaming interactive video e-commerce applications, targeted in-picture advertising using video objects and streaming video.

In addition, the invention allows the streaming of live traffic video to users. This can be performed in a number of alternative ways including where the user dials a special phone number and then selects the traffic camera location to view within the region handled by the operator/exchange, or where a user dials a special phone number and the user's geographic location (derived from GPS or cell triangulation) is used to automatically provide a selection of traffic camera locations to view. Another alternative exists where the user can register for a special service where the service provider will call the user and automatically stream video showing the motorists route that may have a potential traffic jam. Upon registering the user may elect to nominate a route for this device comes into range of a local wireless network (this may be an IEEE 802.11 or bluetooth, etc. type of network), it detects a carrier signal and a server connection request.

If acccepted, the client alerts the user by means of an audible alarm or some other method to indicate that it is initiating the transfer; b) if the user has configured a mobile device to accept these connection requests, then the connection is established with the server else the request is rejected; c) the client sends to the server configuration information including device capabilities such as display screen size, memory capacity and CPU speed, device manufacturer/model and operating system; d) the server receives this information and selects the correct data stream to send to the client. If none is suitable then the connection is terminated; e) after the information is transferred the server closes the connection and the client alerts the user to the end of transmission; and f) if the transmission is unduly terminated due to a lost connection before the transmission is completed, the client cleans up any memory used and reinitialises itself for new connection requests.

Statements of the Invention
In accordance with the present invention there is provided a method of generating an object oriented interactive multimedia file, including:
encoding data comprising at least one of video, text, audio, music and/or graphics
elements as a video packet stream, text packet stream, audio packet stream, music
packet stream and/or graphics packet stream respectively;
combining said packet streams into a single self-contained object, said object
containing its own control information;
placing a plurality of said objects in a data stream; and
grouping one or more of said data streams in a single contiguous self-contained scene, said scene including format definition as the initial packet in a sequence of packets.

The present invention also provides a method of mapping in real time from a nonstationary three-dimensional data set onto a single dimension, comprising the stepsof :
pre-computing said data; encoding said mapping;
transmitting the encoded mapping to a client; and
said client applying said mapping to the said data.

The present invention also provides a system for dynamically changing the actual content of a displayed video in an object-oriented interactive video system comprising:
a dynamic media composition process including an interactive multimedia file format including objects containing video, text, audio, music, and/or graphical data wherein at least one of said objects comprises a data stream, at least one of said data streams comprises a scene, at least one of said scenes comprises a file;
a directory data structure for providing file information;
selecting mechanism for allowing the correct combination of objects to be compositedtogether;
a data stream manager for using directory information and knowing the location of said objects based on said directory information; and control mechanism for inserting, deleting, or replacing in real time while being viewed by a user, said objects in said scene and said scenes in said video.

The present invention also provides an object oriented interactive multimedia file, comprising :
a combination of one or more of contiguous self-contained scenes;
each said scene comprising scene format definition as the first packet, and a group of one or more data streams following said first

packet;
each said data stream apart from first data stream containing objects which may be optionally decoded and displayed according to a dynamic media composition process as specified by object control information in said first data stream; and
each said data stream including one or more single self-contained objects and demarcated by an end stream marker; said objects each containing it's own control information and formed by combining packet streams; said packet streams formed by encoding raw interactive multimedia data including at least one or a combination of video, text, audio, music, or graphics elements as a video packet stream, text packet stream, audio packet stream, music packet stream and graphics packet stream respectively.

The present invention also provides a method of providing a voice command operation of a low power device capable of operating in a streaming video system, comprising the following steps:
capturing a user's speech on said device;
compressing said speech;
inserting encoded samples of said compressed speech into user control packets;
sending said compressed speech to a server capable of processing voice commands;
said server performs automatic speech recognition;
said server maps the transcribed speech to a command set;
said system checks whether said command is generated by said user or said server;
if said transcribed command is from said server, said server executes said command;
if said transcribed command is from said user said system forwards said command to said user device; and
said user executes said command.

The present invention also provides an image processing method, comprising the stepof :
generating a colour map based on colours of an image;
determining a representation of the image using the colour map; and
determining a relative motion of at least a section of the image which is represented using the colour map.

The present invention also provides a method of determining an encoded representation of
an image comprising: analyzing a number of bits utilized to represent a colour;
representing the colour utilizing a first flag value and a first predetermined number of bits, when the number of bits utilized to represent the colour exceeds a first value; and
representing the colour utilizing a second flag value and a second predetermined number of bits, when the number of bits utilized to represent the colour does not exceed a first value.

The present invention also provides an image processing system, comprising means for generating a colour map based on colours of an image;
means for determining a representation of the image using the colour map; and
means for determining a relative motion of at least a section of the image which is represented using the colour map.

The present invention also provides an image encoding system for determining an encoded representation of an image comprising:
means for analyzing a number of bits utilized to represent a colour;
means for representing the colour utilizing a first flag value and a first predetermined number of bits, when the number of bits utilized to represent the colour exceeds a first value; and
means for representing the colour utilizing a second flag value and a second predetermined number of bits, when the number of bits utilized to represent the colour does not exceed a first value.

The present invention also provides a method of processing objects, comprising the stepsof :
parsing information in a script language;
reading a plurality of data sources containing a plurality of objects in the form of at least one of video, graphics, animation, and audio;
attaching control information to the plurality of objects based on the information in the script language; and
interleaving the plurality of objects into at least one of a data stream and a file.

The present invention also provides a system for processing objects, comprising:
means for parsing information in a script language;
means for reading a plurality of data sources containing a plurality of objects in the form of at least one of video, graphics, animation, and audio;
means for attaching control information to the plurality of objects based on the information in the script language; and
means for interleaving the plurality of objects into at least one of a data stream and a file.

The present invention also provides a method of remotely controlling a computer, comprising the stepof :
performing a computing operation at a server based on data;
generating image information at the server based on the computing operation;
transmitting, via a wireless connection, the image information from the server to a client computing device without transmitting said data;
receiving the image information by the client computing device; and
displaying the image information by the client computing device.

The present invention also provides a system for remotely controlling a computer, comprising:
means for performing a computing operation at a server based on data;
means for generating image information at the server based on the computing operation;
means for transmitting, via a wireless connection, the image information from the server to a client computing device without transmitting said data;
means for receiving the image information by the client computing device; and means for displaying the image information by the client computing device.

The present invention also provides a method of transmitting an electronic greeting card, comprising the stepsof :
inputting information indicating features of a greeting card;
generating image information corresponding to the greeting card;
encoding the image information as an object having control information;
transmitting the object having the control information over a wireless connection;
receiving the object having the control information by a wireless hand-held computingdevice;
decoding the object having the control information into a greeting card image by the wireless hand-held computing device; and
displaying the greeting card image which has been decoded on the hand-held computing device.

The present invention also provides a system transmitting an electronic greeting card, comprising:
means for inputting information indicating features of a greeting card;
means for generating image information corresponding to the greeting card;
means for encoding the image information as an object having control information;
means for transmitting the object having the control information over a wireless connection;
means for receiving the object having the control information by a wireless handheld computing device;
means for decoding the object having the control information into a greeting card image by the wireless hand-held computing device; and
means for displaying the greeting card image which has been decoded on the handheld computing device.

The present invention also provides a method of controlling a computing device, comprising the stepsof :
inputting an audio signal by a computing device;
encoding the audio signal;
transmitting the audio signal to a remote computing device;
interpreting the audio signal at the remote computing device and generating information corresponding to the audio signal;
transmitting the information corresponding to the audio signal to the computing device;
controlling the computing device using the information corresponding to the audio signal.

The present invention also provides a system for controlling a computing device, comprising :
inputting an audio signal by a computing device;
encoding the audio signal;
transmitting the audio signal to a remote computing device;
interpreting the audio signal at the remote computing device and generating information corresponding to the audio signal;
transmitting the information corresponding to the audio signal to the computing device; and
controlling the computing device using the information corresponding to the audio signal.

The present invention also provides a system for performing a transmission, comprising:
means for displaying an advertisement on a wireless hand-held device;
means for transmitting information from the wireless hand-held device; and
means for receiving a discounted price associated with the information which has been transmitted because of the display of the advertisement.

The present invention also provides a method of providing video, comprising the stepsof :
determining whether an event has occurred; and
obtaining a video of an area transmitting to a user by a wireless transmission the video of the area in response to the event.

The present invention also provides a system for providing video, comprising:
means for determining whether an event has occurred;
means for obtaining a video of an area; and
means for transmitting to a user by a wireless transmission the video of the area in response to the event.

The present invention also provides an object oriented multimedia video system capable of supporting multiple arbitrary shaped video objects without the need for extra data overhead or processing overhead to provide video object shape information.

The present invention also provides a method of delivering multimedia content to wireless devices by server initiated communications wherein content is scheduled for delivery at a desired time or cost effective manner and said user is alerted to completion of delivery via device's display or other indicator.

The present invention also provides an interactive system wherein stored information can be viewed offline and stores user input and interaction to be automatically forwarded over a wireless network to a specified remote server when said device next connects online.

The present invention also provides a video encoding method, including:
encoding video data with object control data as a video object; and
generating a data stream including a plurality of said video object with respective video data and object control data.

The present invention also provides a video encoding method, including:
quantising colour data in a video stream based on a reduced representation of colours;
generating encoded video frame data representing said quantised colours and transparent regions; and
generating encoded audio data and object control data for transmission with said encoded video data.

The present invention also provides a video encoding method, including:
(i) selecting a reduced set of colours for each video frame of video data;
(ii) reconciling colours from frame to frame;
(iii) executing motion compensation;
(iv) determining update areas of a frame based on a perceptual colour difference measure;
(v) encoding video data for said frames into video objects based on steps (i) to (iv); and
(vi) including in each video object animation, rendering and dynamic composition controls.

The present invention also provides a wireless streaming video and animation system, including:
(i) a portable monitor device and first wireless communication means;
(ii) a server for storing compressed digital video and computer animations and enabling a user to browse and select digital video to view from a library of available videos; and
(iii) at least one interface module incorporating a second wireless communication means for transmission of transmittable data from the server to the portable monitor device, the portable monitor device including means for receiving said transmittable data, converting the transmittable data to video images displaying the video images, and permitting the user to communicate with the server to interactively browse and select a video to view.

The present invention also provides a method of providing wireless streaming of video and animation including at least one of the stepsof :
(a) downloading and storing compressed video and animation data from a remote server over a wide area network for later transmission from a local server;
(b) permitting a user to browse and select digital video data to view from a library of video data stored on the local server;
(c) transmitting the data to a portable monitor device; and
(d) processing the data to display the image on the portable monitor device.

The present invention also provides a method of providing an interactive video brochure including at least one of the stepsof :
(a) creating a video brochure by specifying (i) the various scenes in the brochure and the various video objects that may occur within each scene, (ii) specifying the preset and user selectable scene navigational controls and the individual composition rules for each scene, (iii) specifying rendering parameters on media objects, (iv) specifying controls on media objects to create forms to collect user feedback, (v) integrating the compressed media streams and object control information into a composite data stream.

The present invention also provides a method of creating and sending video greeting cards to mobile devices including at least one of the stepsof :
(a) permitting a customer to create the video greeting card by (i) selecting a template video scene or animation form a library, (ii) customising the template by adding user supplied text or audio objects or selecting video objects from a library to be inserted as actors in the scene;
(b) obtaining from the customer (i) identification details, (ii) preferred delivery method, (iii) payment details, (iv) the intended recipient's mobile device number; and
(c) queuing the greeting card depending on the nominated delivery method until either bandwidth becomes available or off peak transport can be obtained, polling the recipient's device to see if it is capable of processing the greeting card and if so forwarding to the nominated mobile device.

The present invention also provides a video decoding method for decoding the encoded data.

The present invention also provides a dynamic colour space encoding method to permit further colour quantisation information to be sent to the client to enable real-time client based colour reduction.

The present invention also provides a method of including targeted user and/or local video advertising.

The present invention also includes executing an ultrathin client, which may be wireless, and which is able to provide access to remote servers.

The present invention also provides a method for multivideo conferencing.

The present invention also provides a method for dynamic media composition.

The present invention also provides a method for permitting users to customise and forward electronic greeting cards and post cards to mobile smart phones.

The present invention also provides a method for error correction for wireless streaming of multimedia data.

The present invention also provides systems for executing any one of the above methods, respectively.

The present invention also provides server software for permitting users to a method for error correction for wireless streaming of video data.

The present invention also provides a computer software for executing steps of any one of the above methods, respectively.

The present invention also provides a video on demand system. The present invention also provides a video security system. The present invention also provides an interactive mobile video system.

The present invention also provides a method of processing spoken voice commands to control the video display.

The present invention also provides software including code for controlling object oriented video and/or audio. Advantageously, the code may include IAVML instructions, why may be based on XML.

Brief Description of Drawings
Preferred embodiments of the present invention are hereinafter described, by way of example only, with reference to the accompanying drawings, wherein:
Figure 1 is a simplified block diagram of an object oriented multimedia system of one embodiment of the present invention;
Figure 2 is a schematic diagram illustrating the three major packet types interleaved into an object oriented data stream of the embodiment illustrated in
Figure 1;
Figure 3 is a block diagram illustrating the three phases of data processing in an object oriented multimedia player embodiment of the present invention;
Figure 4 is a schematic diagram showing the hierarchy of object types in an object oriented data file according to the present invention;
Figure 5 is a diagram showing a typical packet sequence in a data file or stream according to the present invention;
Figure 6 is a diagram illustrating the information flow between client and server components of an object oriented multimedia player according to the present invention;
Figure 7 is a block diagram showing the major components of an object oriented multimedia player client according to the present invention;
Figure 8 is a block diagram showing the functional components of an object oriented multimedia player client according to the present invention;
Figure 9 is a flow chart describing the major steps in the multi-object client rending process according to the present invention;
Figure 10 is a block diagram of a preferred embodiment of the client rendering engine according to the present invention;
Figure 11 is a block diagram of a preferred embodiment of the client interaction engine according to the present invention;
Figure 12 is a component diagram describing an embodiment of an interactive multi-object video scene with DMC functionality.

Figure 13 is a flow chart describing the major steps in the process the client performs in playing an interactive object oriented video according to the present invention;
Figure 14 is a block diagram of the local server component of an interactive multimedia player according to the present invention;
Figure 15 is a block diagram of a remote streaming server according to the present invention;
Figure16 Is a flow chart describing the main steps executed by a client performing dynamic media composition according to the present invention;
Figure17 Is a flow chart describing the main steps executed by a server client performing dynamic media composition according to the present invention;
Figure 18 is a block diagram of an object-oriented video encoder according to the present invention;
Figure 19 is a flow chart of the main steps executed by a video encoder according to the present invention;
Figure 20 is a block diagram of an input colour processing component of a video encoder according to the present invention;
Figure 21 is a block diagram of the components of a region update selection process used in a video encoder according to the present invention;
Figure 22 is a diagram of three fast motion compensation methods used in video encoding;
Figure 23 is a diagram of the tree splitting method used in a video encoder according to the present invention;
Figure 24 is a flow chart of the main stages performed to encode the data resulting from the video compression process according to the present invention;
Figure 25 is a flow chart of the steps for encoding the colour map update information according to the present invention;
Figure 26 is a flow chart of the steps to encode the quad tree structure data for normal predicted frames according to the present invention;
Figure 27 is a flow chart of the steps to encode the leaf colour in the quad tree data structure according to the present invention;
Figure 28 is a flow chart of the main steps executed by a video encoder to compress video key frames according to the present

invention;
Figure 29 is a flow chart of the main steps executed by a video encoder to compress video using the alternate encoding method according to the present invention;
Figure 30 is a flow chart of the main involved in the prequantisation process to perform real-time colour (vector) quantisation in real-time at the client according to the present invention;
Figure 31 is a flow chart of the main steps in the voice command process according to the present invention;
Figure 32 is a block diagram of an ultra-thin computing client Local Area wireless Network (LAN) system according to the present invention;
Figure 33 is a block diagram of an ultra-thin computing client Wide Area wirelessNetwork (WAN) system according to the present invention;
Figure 34 is a block diagram of an ultra-thin computing client Remote LAN server system according to the present invention;
Figure 35 is a block diagram of an multiparty wireless videoconferencing system according to the present invention;
Figure 36 is a block diagram of one embodiment of an interactive'video on demand'system, with targeted in-picture user advertising, according to the present invention;
Figure 37 is a flow chart of the main steps involved in the process of delivering and handling one embodiment of an interactive in-picture targeted user advertisement according to the present invention;
Figure 38 is a flow chart of the main steps involved in the process of playing and handling one embodiment of an interactive video brochure according to the present invention;
Figure 39 is a flow chart of a sequence of possible user interactions in one embodiment of an interactive video brochure according to the present invention;
Figure 40 is a flow chart of the main steps involved in push or pull based distribution of video data according to the present invention;
Figure 41 is a block diagram of an interactive'video on demand'system according to the present invention, with remote server based digital rights management functions including user authentication, access control, billing and usage metering;
Figure 42 is a flow chart of the main steps of the process that player software performs in playing on demand streaming wireless video according to the present invention;
Figure 43 is a block diagram of a videosecurity/surveillance systems according to the present invention
Figure 44 is a block diagram of an electronic greeting card system and service according to the present invention.

Figure 45 is a flow chart of the main steps involved in creating and sending a personalised electronic video greeting card or videoE-mail to a mobile telephone according to the present invention;
Figure 46 is a block diagram showing the centralised parametric scene description used in the MPEG4 standard;
Figure 47 is a block diagram showing the main steps in providing colour quantisation data to a decoder for real time colour quantisation according to the present invention;
Figure 48 is a block diagram showing the main components of an object library according to the present invention;
Figure 49 is a flowchart of the main steps of a video decoder according to the present invention;
Figure 50 is a flowchart of the main steps involved in decoding a quad tree encoded video frame according to the present invention.

Figure 51 is a flowchart of the main steps involved in decoding a leaf colour of a quad tree according to the present invention.

Detailed Description of the Invention
Glossary of Terms
Bit Stream A sequence of bits transmitted from a server to a client, but may be stored in memory.

Data Stream One or more interleaved Packet Streams.

Dynamic Media Composition Changing the composition of a multi-object multimedia presentation in real time.

File An object oriented multimedia file.

In Picture Object An overlayed video object within a scene.

Media Object A combination of one or more interleaved media types including audio, video, vector graphics, text and music.

Object A combination of one or more interleaved media types including audio, video, vector graphics, text and music.

Packet Stream A sequence of data packets belonging to one object transmitted from a server to a client but may be stored in memory.

Scene The encapsulation of one or more Streams, comprising a multi-object multimedia presentation.

Stream A combination of one or more interleaved Packet Streams, stored in an object oriented multimedia file.

Video Object A combination of one or more interleaved media types including audio, video, vector graphics, text and music.

Acronyms
The following acronyms are used herein:
FIFO First In First Out Buffer.

IAVML Interactive Audio Visual Mark-up Language
PDA Personal Digital Assistant
DMC Dynamic Media Composition
IME Interaction Management Engine
DRM Digital Rights Management
ASR Automatic Speech Recognition
PCMCIA Personal Computer Memory Card International Association General System Architecture
The processes and algorithms described herein form an enabling technology platform for advanced interactive rich media applications such as E-commerce. The great advantage of the methods described is that they can be executed on very low processing power devices such as mobile phones and PDAs in software only, if desired. This will become more apparent from the flow chart and accompanying descriptions as shown in Figure 42. The specified video codec isfun number of encoding phases together with definition and control data according to a given script, and sends the resulting data stream to the player client. The player client includes a decoding engine 62, which decompresses the object data stream and renders the various objects before sending them to the appropriate hardware output devices61.

Referring to Figure 2, the decoding engine 62 performs operations on three interleaved streams of data: compressed data packets 64, definition packets 66, and object control packets 68. The compressed data packets 64 contain the compressed object (e.g., video) data to be decoded by an applicable encoder/decoder ('codec'). The methods for encoding and decoding video data are discussed in a later section. The definition packets 66 convey media format and other information that is used to interpret the compressed data packets 64. The object control packets 68 define object behaviour, rendering, animation and interaction parameters.

Figure 3 is a block diagram illustrating the three phases of data processing in an object oriented multimedia player. As shown, three separate transforms are applied to the object oriented data to generate a final audio-visual presentation via a system display 70 and an audio subsystem. A'dynamic media composition' (DMC) process 76 modifies the actual content of the data stream and sends this to the decoding engine 62. In the decoding engine 62, a normal decoding process 72 extracts the compressed audio and video data and sends it to a rendering engine 74 where other transformations are applied, including geometric transformations of rendering parameters for individual objects,(e. g., translation). Each transformation is individually controlled through parameters inserted into the data stream.

The specific nature of each of the final two transformations depends on the output of the dynamic media composition process 76, as this determines the content of the data stream passed to the decoding engine 62. For example, the dynamic media composition process 76 may insert a specific video object into the bit stream. In this case, in addition to the video data to be decoded, the data bit stream will contain configuration parameters for the decoding process 72 and the rendering engine 74.

The object oriented bit stream data format permits seamless integration between different kinds of media objects, supports user interaction with these objects, and enables programmable control of the content in a displayed scene, whether streaming the data from a remote server or accessing locally stored content.

Figure 4 is a schematic diagram showing the hierarchy of object types in an object oriented multimedia data file. The data format defines a hierarchy of entities as follows: an object oriented data file 80 may contain one or more scenes 81. Each scene may contain one or more streams 82 which contain one or more separate simultaneous media objects 52. The media objects 52 may be of a single media element 89 such as video 83, audio 84, text 85, vector graphics (GRAF) 86, music 87 or composites of such elements 89. Multiple instances of each of the above said media types may simultaneously occur together with other media types in a single scene. Each object 52 can contain one or more frames 88 encapsulated within data packets. When more than one media object 52 is present in a scene 81, the packets for each are interleaved. A single media object 52 is a totally selfcontained entity that has virtually no dependencies. It is defined by a sequence of packets including one or more definition packets 66, followed by data packets 64 and any control packets 68 all bearing the same object identifier number. All packets in the data file have the same header information (the baseheader) which specifies the object that the packet corresponds to, the type of data in the packet, the number of the packet in a sequence and the amount of data (size) the packet contains. Further details of the file format are described in a later section.

The distinction with the MPEG4 system will be readily observed. Refering to Figure 46, MPEG4 relies on a centralised parametric scene description in the form of the Binary Format for Scenes (BIFS) Ola, which is a hierarchical structure of nodes that can contain the attributes of objects and other information. BIFS Ola is borrowed directly from the very complex Virtual Reality Markup Language (VRML) Grammar. In this approach, the centralised BIFS structure Ola is actually the sceneitself : it is the fundamental component in an object oriented video, not the objects themselves. Video object data may be specifed for use in a scene, but does not serve in defining the scene itself. So, for example, a new video object cannot be introduced into a scene unless the BIFS structure Ola is first modified to include a node that references the video data. The BIFS also does not directly reference any object data streams; instead, a special intermediary independent device called an object descriptorOlb maps between any OBJIDs in the nodes of a BIFS Ola and the elementary data streams Olc which contain video data. Hence in the MPEG approach each of these three separate entitiesOla, Olb, Olc, are interdependent, so that if an object stream is copied to another file, it loses any interactive behaviour and any other control information associated with it. Since MPEG4 is not object-centric, its data packets are referred to as atoms which have a common header consisting of only type and packet size information, but no object identifier.

The format described herein is much simpler, since there is no central structure that defines what the scene is. Instead, the scene is self-contained and completely defined by the objects that inhabit the scene. Each object is also self-contained, having attached any control information that specifies the attributes and interactive behaviour of the object.

New objects can be copied into a scene just by inserting their data into the bitstream, doing this introduces all of the objects'control information into the scene as well as their compressed data. There are virually no interdependencies between media objects or between scenes. This approach reduces the complexity and the storage and processing overheads associated with the complex BIFs approach.

In the case of download and play of video data, to allow interactive, object oriented manipulation of multimedia data, such as the ability to choose which actors appear in a scene, the input data does not include a single scene with a single"actor"object, but rather one or more alternative object data streams within each scene that may be selected or "composited-in"to the scene displayed at run-time, based on user input. Since the composition of the scene is not known prior to runtime, it is not possible to interleave the correct object data streams into the scene.

Figure 5 is a diagram showing a typical packet sequence in a data file. A stored scene 81 includes a number of separate selectable streams 82, one for each"actor"object 52 that is a candidate for the dynamic media composition process 76, referred to in Figure 3. Only the first stream 82 in a scene 81 contains more than one (interleaved) media object 52. The first stream 82 within a scene 81 defines the scene structure, the constituent objects and their behaviour. Additional streams 82 in a scene 81 contain optional object data streams 52. A directory 59 of streams is provided at the beginning of each scene 81 to enable random access to each separate stream 82.

While the bit stream is capable of supporting advanced interactive video capabilities and dynamic media composition, it supports three implementation levels, providing various levels of functionality. These are: 1. Passive media: Single-object, non-interactive player 2. Interactive media: Single-object, limited interaction player 3. Object-oriented active media: Multi-object, fully interactive player
The simplest implementation provides a passive viewing experience with a single instance of media and no interactivity. This is the classic media player where the user is limited to playing, pausing and stopping the playback of normal video or audio.

The next implementation level adds interaction support to passive media by permitting the definition of hot regions for click-through behaviour. This is provided by creating vector graphic objects with limited object control functionality. Hence the system is not literally a single object system, although it would appear so to the user. Apart from the main media object being viewed transparent, clickable vector graphic objects are the other types of objects permitted. This allows simple interactive experiences to be created such as nonlinear navigation, etc.

The final implementation level defines the unrestricted use of multiple objects and full object control functionality, including animations, conditional events, etc., and uses the implementation of all of the components in this architecture. In practice, the differences between this level and the previous may only be cosmetic.

Figure 6 is a diagram illustrating the information flow (or bit stream) between client and server components of an object-oriented multimedia system. The bit stream supports client side and server side interaction. Client side interaction is supported via a set of defined actions that may be invoked through objects that cause modification of the user experience, shown herein as object control packets 68. Server side interaction support is where user interaction, shown here as user control packets 69, is relayed from a client 20 to a remote server 21 via a back channel, and provides mediation of the service/content provision to online users, predominantly in the form of dynamic media composition.

Hence an interactive media player to handle the bit stream has a client-server architecture.

The client 20 is responsible for decoding compressed data packets 64, definition packets 66 and object control packets 68 sent to it from the server 21. Additionally the client 20 is responsible for object synchronisation, applying the rendering transformations, compositing the final display output, managing user input and forwarding user control back to the server 21. The server 21 is responsible for managing, reading, and parsing partial bit streams from the correct source (s), constructing a composite bit stream based on user input with appropriate control instructions from the client 20, and forwarding the bit stream to the client 20 for decoding and rendering. This server side Dynamic Media
Composition, illustrated as component 76 of Figure 3, permits the content of the media to be composited in real-time, based on user interaction or predefined settings in a stored program script.

The media player supports both server side and client sideinteraction/functionality when playing back data stored locally, and also when the data is being streamed from a remote server 21. Since it is the responsibility of the server component 21 to perform the DMC and manage sources, in the local playback case the server is co-located with the client 20, while being remotely located in the streaming case. Hybrid operation is also supported, where the client 20 accesses data from local and remotely located source/servers21.

Interactive Client
Figure 7 is a block diagram showing the major components of an object oriented multimedia player client 20. The object oriented multimedia player client 20 is able to receive and decode the data transmitted by the server 21 and generated by the DMC process 76 of Figure 3. The object oriented multimedia player client 20 also includes a number of components to execute the decoding process. The steps of the decoding process are simplistic when compared to the encoding process, and can be executed entirely by software compiled on a low power mobile computing device such as a Palm
PilotIIlc or a smart phone. An input data buffer 30 is used to hold the incoming data from the server 21 until a full packet has been received or read. The data is then forwarded to an input data switch/demux 32, either directly or via a decryption unit 34. The input dataswitch/demux 32 determines which of sub-processes 33,38,40,42 is required to decode the data, and then forwards the data to the correct component according to the packet type that executes that sub-process. Separate components 33,38 and 42 perform vector graphics, video, and audio decoding respectively. The video and audio decoding modules 38 and 42 in the decoder independently decompress any data sent to them and perform a preliminary rendering into a temporary buffer. An object management component 40 extracts object behaviour and rendering information for use in controlling the video scene.

A video display component 44 renders visual objects on the basis of data received from the vector graphics decoder 33, video decoder 38 and the object management component 40. An audio play back component 46 generates audio on the basis of data received from the audio decoding and object management component 40. A user input/control component 48 generates instructions and controls the video and audio generated by the display and playback components 44 and 46. The user control component 48 also transmits control messages back to the server21.

Figure 8 is a block diagram showing thefunctional components of an object oriented multimedia player client 20, including the following:
1. Decoders 43 with optional object stores 39 for the main data paths (a combination
of a plurality of components 33,38 and 42 of Figure 7)
2. Rendering engine 74 (components 44 and 46 of Figure 7 combined)
3. Interaction management engine 41 (components 40 and 48 of Figure 7 combined)
4. Object control 40 path (part of component 40 of Figure 7)
5. Input data buffer 30 and input data switch/demux 32.

6. Optional digital rights management (DRM) engine 45
7. Persistent local object library 75

There are two principle flows of data through the client system 20. Compressed object data 52 is delivered to the client input buffer 30 from the server 21 or the persistent local object library 75. The input data switch/demux 32 splits up the buffered compressed object data 52 into compressed data packets 64, definition packets 66 and object control packets 68. Compressed data packets 64 and definition packets 66 are individually routed to the appropriate decoder 43 based on the packet type as identified in the packet header.

Object control packets 68 are sent to the object control component 40 to be decoded.

Alternatively, the compressed data packets 64, definition packets 66 and object control packets 68 may be routed from the input dataswitch/demux 32 to the object library 75 for persistent local storage, if an object control packet is received specifying library update information. One decoder instance 43 and object store 39 exists for each media object and for each media type. Hence there are not only different decoders 43 for each media type, but if there are three video objects in a scene, then there will be three instances of video decoders 43. Each decoder 43 accepts the appropriate compressed data packets 64 and definition packets 66 sent to it and buffers the decoded data in the object data stores 39.

Each object store 39 is responsible for managing the synchronisation of each media object in conjunction with the rendering engine 74; if the decoding is lagging the (video) frame refresh rate, then the decoder 43 is instructed to drop frames as appropriate. The data in the object stores 39 is read by the rendering engine 74 to compose the final displayed scene. Read and write access to the object data stores 39 is asynchronous such that the decoder 43 may only update the object data store 39 at a slow rate, while the rendering engine 74 may be reading that data at a faster rate, or vice versa, depending on the overall media synchronisation requirements. The rendering engine 74 reads the data from each of
the object stores 39 and composes both the final display scene and the acoustic scene, based on rendering information from the interaction management engine 41. The result of this process is a series of bitmaps that are handed over to the system graphical user interface 73 to be displayed on the display device 70 and a series of audio samples to be passed to the system audio device 72.

The secondary data flow through the client system 20 comes from the user via the graphical user interface 73, in the form of User Events 47, to the interaction management engine 41, where the user events are split up, with some of them being passed to the rendering engine 74 in the form of rendering parameters, and the rest being passed back through a back channel to the server 21 as user control packets 69; the server 21 uses these to control the dynamic media composition engine 76. To decide where or if user events are to passed to other components of the system, the interaction management engine 41 may request the rendering engine 74 to perform hit testing. The operation of the interaction management engine 41 is controlled by the object control component 40, which receives instructions (object control packets 68) sent from the server 21 that define how the interaction management engine 41 interprets user events 47 from the graphical user interface 73, and what animations and interactive behaviours are associated with individual media objects. The interaction management engine 41 is responsible for controlling the rendering engine 74 to carry out the rendering transformations.

Additionally, the interaction management engine 41 is responsible for controlling the object library 75 to route library objects into the input data switch/demux 32.

The rendering engine 74 has four main components as shown in Figure 10. A bitmap compositor 35 reads bitmaps from the visual object store buffers 53 and composites them into the final display scene raster 71. A vector graphic primitive scan converter 36 renders the vector graphic display list 54 from the vector graphic decoder onto the display scene raster 71. An audio mixer 37 reads the audio object stores 55 and mixes the audio data together before passing the result to the audio device 72. The sequence in which the various object store buffers 53 to 55 are read and how their content is transformed onto the display scene raster 71 is determined by rendering parameters 56 from the interaction management engine 41. Possible transformations include Z-order, 3D orientation, position, scale, transparency, colour, and volume. To speed up the rendering process, it may not be necessary to render the entire display scene, but only a portion of it. The fourth main component of the rendering engine is the Hit Tester31, which performs object hit testing for user pen

events as directed by the user event controller 41c of the interaction management engine41.

The display scene should be rendered whenever visual data is received from the server 21 according to synchronization information, when a user selects a button by clicking or drags an object that is draggable, and when animations are updated. To render the scene, it may be composited into an offscreen buffer (the display scene raster 71), and then drawn to the output device 70. The object rendering/bitmap compositing process is shown in
Figure 9, beginning at step s101. A list is maintained that contains a pointer to each media object store containing visual objects. The list is sorted according to Z order at steps102. Subsequently, at steps103, the bitmap compositer gets the media object with the lowest Z order. If at steps104 there are no further objects to composite, the video object rendering process ends at steps118. Otherwise, and always in the case of the first object, the decoded bitmap is read from the object buffer at steps105. If, at steps106, there are object rendering controls, then the screen position, orientation and scale are set at steps107. Specifically, the object rendering controls define the appropriate 2/3D geometric transform to determine which coordinates the object pixels are mapped to. The first pixel is read from the object buffer at stepss108, and, if there are more pixels to process ats109, reads the next pixel from the object buffer at steps110. Each pixel in the object buffer is processed individually. If, at stepsl l l, the pixel is transparent (pixel value isOxFE), then the rendering process ignores the pixel and returns to steps109 to begin processing the next pixel in the object buffer. Otherwise, if the pixel is unchanged (pixel value is OxFF) at steps112, then a background colour pixel is drawn to the display scene raster at steps113. However, if the pixel is neithier transparent nor unchanged, and alpha blending is not enabled at stepsl 14, the object colour pixel is drawn to the display scene raster at step
sol5. If alpha blending is enabled at steps 114, then an alpha blending composition process is performed to set the defined level of transparency for the object. However, unlike traditional alpha blending processes that need to separately encode the mixing factor for every pixel in a bitmap, this approach does not make use of an alpha channel.

Instead, it utilizes a single alpha value specifying the degree of opacity of the entire bitmap in conjunction with embedded indication of transparent regions in the actual bitmap representation. Thus, when the new alpha blending object pixel colour is calculated at steps116, it is drawn to the display scene raster at stepsl 17. This concludes the processing for each individual pixel, thus control returns to steps109, to begin processing the next pixel in the object buffer. If no pixels remain to be processed at steps109, the process returns to steps104 to begin processing the next object. The bitmap compositor 35 reads each video object store in sequence according to the Z-order associated with each media object, and copies it to the display scene raster 71. If no Z order has been explicitly assigned to objects, the z order value for an object can be taken to be the same as theobject ID. If two objects have the same Z order, they are drawn in order of ascending object IDs.

As described, the bitmap compositor 35 makes use of the three region types that a video frame can have: colour pixels to be rendered, areas to be made transparent, and areas to remain unchanged. The colour pixels are appropriately alpha blended into the display scene raster 71, and the unchanged pixels are ignored so the display scene raster 71 is unaffected. The transparent pixels force the corresponding background display scene pixel to be refreshed. This can be performed when the pixel of the object in question is overlaying some other object by simply doing nothing, but if the pixel is being drawn directly over the scene background, then that pixel needs to be set to the scene background colour.

If the object store contains a display list in place of a bitmap, then the geometric transform is applied to each of the coordinates in the display list, and the alpha blending is performed during the scan conversion of the graphics primitives specified within the display list.

Refering to Figure 10, the bitmap compositor 35 supports display scene rasters with different colour resolutions, and manages bitmaps with different bit depths. If the display scene raster 71 has a depth of 15,16 or 24 bits, and a bitmap is a colour mapped 8 bit image, then the bitmap compositor 35 reads each colour index value from the bitmap, looks up the colour in the colour map associated with that particular object store, and writes the red, green and blue components of the colour in the correct format to the display scene raster71. If the bitmap is a continuous tone image, the bitmap compositor 35 simply copies the colour value of each pixel into the correct location on the display scene raster 71. If the display scene raster 71 has a depth of 8 bits and a colour look up table, the approach taken depends on the number of objects displayed. If only one video object is being displayed, then its colour map is copied directly into the colour map of the display scene raster 71. If multiple video objects exist, then the display scene raster 71 will be set up with a generic colour map, and the pixel value set in the display scene raster 71 will be the closest match to the colour indicated by the index value in the bitmap.

The hit tester component 31 of the rendering engine 74 is responsible for evaluating when a user has selected a visual object on the screen by comparing the pen event location coordinates with each object displayed. This'hit testing'is requested by the user event controller 41c of the interaction management engine 41, as shown in Figure 10, and utilizes object positioning and transformation information provided by the bitmap compositor 35 and vector graphic primitive scan convertor 36 components. The hit tester 31 applies an inverse geometric transformation of the pen event location for each object, and then evaluates the transparency of the bitmap at the resulting inverse-transformed coordinate. If the evaluation is true, a hit is registered, and the result is returned to the user event controller41 c of the interaction management engine41.

The rendering engines'audio mixer component 37 reads each audio frame stored in the relevant audio object store in round-robin fashion, and mixes the audio data together according to the rendering parameters 56 provided by the interaction engine to obtain the composite frame. For example, a rendering parameter for audio mixing may include volume control. The audio mixer component 37 then passes the mixed audio data to the audio output device 72.

The object control component 40 of Figure 8 is basically a codec that reads the coded object control packets from the switch/demux input stream and issues the indicated control instructions to the interaction management engine 41. Control instructions may be issued to change individual objects or system wide attributes. These controls are wideranging, and include rendering parameters, definition of animation paths, creating conditional events, controlling the sequence of media play including inserting objects from the object library 75, assigning hyperlinks, setting timers, setting and resetting system state registers, etc, and defining user-activated object behaviours.

The interaction engine 41 has to manage a number of different processes; the flowchart of
Figure 13 shows the major steps an interactive client performs in playing an interactive object oriented video. The process begins at step s201. Data packets and control packets are read at step s202 from the input data source, either the Object Stores 39 of Figure 8, or the Object Control component 40 of Figure 8. If, at step s203, the packet is a data packet, the frame is decoded and buffered at step s204. If, however, the packet is an object control packet, the interaction engine 41 attaches the appropriate action to the object at step s206. The object is then rendered at step s205. If, at step s207, there has been no user interaction with an object (i. e. user has not clicked on the object), and, at step s208, no objects have waiting actions, then the process returns to step s202, and a new packet is read from the input data source at step s202. However, if at step s208, the object has waiting actions, or if there was no user interaction, but the object has an attached action at step s209, the object action conditions are tested at steps210, and if
ObjectControl packets 68, as shown in Figure 8. The interaction engine 41 may immediately perform predefined actions unconditionally (such as jumping back to the start of a scene when the last video frame in the scene is reached), or delay execution until some system conditions are met (such as a timer event occurring), or it may respond to user input (such as clicking or dragging an object) with a defined behaviour, either unconditionally, or subject to system conditions. Possible actions include rendering attribute changes, animations, looping and non-sequential play sequences, jumping to hyperlinks, dynamic media composition where a displayed object stream is replaced by another object, possibly from the persistent local object library 75, and other system behaviours that are invoked when given conditions or user events become true

The interaction management engine 41 includes three main components: an interaction control component4 la, a waiting actions manager4 1 d, and an animation manager 41b, as shown in Figure 11. The animation manager 41b includes the Interaction Control component41a and the Animation Path Interpolator/Animation List 41b, and stores all animations that are currently in progress. For each active animation, the manager interpolates the rendering parameters 56 sent to the rendering engine 74 at intervals specified by the object control logic 63. When an animation has completed, it is removed from the list of active animations, the Animation list 41b, unless

it is defined to be a looping animation. The waiting actions manager41d includes the Interaction Control component41d and the Waiting Actions List41d, and stores all object control actions to be applied subject to a condition becoming true. The interaction control component41a regularly polls the waiting actions manager41d and evaluates the conditions associated with each waiting action. If the conditions for an action are met, the interaction control component41a will execute the action and purge it from the waiting actions list41d, unless the action has been defined as an object behaviour, in which case it remains on the waiting actions list41d for further future executions. For condition evaluation, the interaction management engine 41 employs a condition evaluator41f, and a state flags register41e. The state flags register41e is updated by the interaction control component41a, and maintains a set of user-definable system flags. The condition evaluator41f performs condition evaluation as instructed by the interaction control component41a, comparing the current system state to the system flags in the state flags register41e on a per object basis, and if the appropriate system flags are set, the condition evaluator41f notifies the interaction control component41 a that the condition is true, and that the action should be executed. If the client is offline (i. e., not connected to a remote server), the interaction control component41a maintains a record of all interaction activities performed (user events, etc). These are temporarily stored in the history/form store41d and are sent to the server using user control packets 69 when the client comes online.

Object control packets 68 and hence the object control logic 63 may set a number of userdefinable system flags. These are used to permit the system to have a memory of its current state, and are stored in the state flags register41e. For example, one of these flags may be set when a certain scene or frame in the video is played, or when a user interacts with an object. User interaction is monitored by the user event controller 41c, receiving as input user events 47 from the grapical user interface 73. Additionally, the user event controller 41c may request the rendering engine 74 to perform'hit testing', using the rendering engines'hit tester31. Typically, hit testing is requested for user pen events, such as user pen click/tap. The user event controller 41c forwards user events to the interaction control component4 la. This may then be used to determine what scene to play next in nonlinear videos, or what objects to render in a scene. In an e-commerce application, the user may drag one or more iconic video objects onto a shopping basket object. This will then register the intended purchases. When the shopping basket is clicked, the video will jump to the checkout scene, where a list of all of the objects that were dragged onto the shopping basket appears, permitting the user toconfirm or delete the items. A separate video object can be used as a button, indicating that the user wishes to register the purchase order or cancel it.

Object control packets 68 and hence the object control logic 63 may contain conditions that is satisfied for any specified actions to be executed; these are evaluated by the condition evaluator41f. Conditions may include the system state, local or streaming playback, system events, specific user interactions with objects, etc. A condition may have the wait flag set, indicating that if the condition isn't currently satisfied, then wait until it is. The wait flag is often used to wait for user events such aspenUp. When a waiting action is satisfied, it is removed from the waiting actions list41d associated with an object.

If the behaviour flag of an Object control packet 68 is set, then the action will remain with an object in the waiting actions list41d, even after it has executed.

An Object control packet 68 and hence the object control logic 63 may specify that the action is to affect another object. In this case, the conditions should be satisfied on the object specified in the base header, but the action is executed on the other object. The object control logic may specify object library controls 58, which are forwarded to the object library 75. For example, the object control logic 63 may specify that a jumpto (hyperlink) action is to be performed together with an animation, with the conditions being that a user click event on the object is required, evaluated by the user event controller 41c in conjunction with the hit tester31, and that the system should wait for this to become true before executing the instruction. In this case, an action or control will wait in the waiting actions list41d until it is executed and then it will be removed. A control like this may, for example, be associated with a pair of running shoes being worn by an actor in a video, so that when users click on them, the shoes may move around the screen and zoom in size for a few seconds before the users are redirected to a video providing sales information for the shoes and an opportunity to purchase or bid for the shoes in an online auction.

Figure 12 illustrates the composition of a multi-object interactive video scene. The final scene 90 includes a background video object 91, three arbitary shape"channel change" video objects 92, and three"channel"video objects 93a, 93b and 93c. An object may be defined as a"channel changer"92 by assigning a control with"behaviour","jumpto"and "other"properties, with a condition of user click event. This control is stored in the waiting actions list41d until the end of the scene occurs and will cause the DMC to change the composition of the scene 90 whenever it is clicked. The"channel changing" object in this illustration would display a miniature version of the content being shown on the other channel.

An object control packet 68, and hence the object control logic 63 may have the animation flag set, indicating that multiple commands will follow rather than a single command (such as move to). If the animation flag isn't set, then the actions are executed as soon as the conditions are satisfied. As often as any rendering changes occur, the display scene should be updated. Unlike most rendering actions that are driven by either user events 47 or object control logic 63, animations should force rendering updates themselves. After the animation is updated, and if the entire animation is complete, it is removed from the animation list41b. The animation path interpolator 41b determines where, between which two control points, the animation is currently positioned. This information, along with a ratio of how far the animation has progressed between the two control points(the'tweening'value), is used to interpolate the relevant rendering parameters 56. The tween value is expressed as a ratio in terms of a numerator and denominator:
X = x [start) + (x [end]-x [start]) * numerator/denominator
If the animation is set to loop, then the start time of the animation is set to the current time when the animation has finished, so that it isn't removed after the update.

The client supports the following types of high-level user interaction: clicking, dragging, overlapping, and moving. An object may have a button image associated with it that is displayed when the pen is held down over an object. If the pen is moved a specified number of pixels when it is down over an object, then the object is dragged (as long as dragging isn't protected by the object or scene). Dragging actually moves the object under the pen. When the pen is released, the object is moved to the new position unless moving is protected by the object or scene. If moving is protected, then the dragged object moves back to its original position when the pen is released. Dragging may be enabled so that users can drop objects on top of other objects (e.g., dragging an item onto a shopping basket). If the pen is released whilst the pen is also over other objects, then these objects are notified of an overlap event with the dragged object.

Objects may be protected from clicks, moving, dragging, or changes in transparency or depth through object control packets 68. A PROTECT command within an object control packet 68 may have individual object scope or system scope. If it has system scope, then all objects are affected by the PROTECT command. System scope protection overrides object scope protection.

The JUMPTO command has four variants. One permits jumping to a new given scene in a separate file specified by a hyperlink, another permits replacing a currently playing media object stream in the current scene with another media object from a separate file or scene specified by a hyperlink, and the other two variants permit jumping to a new scene within the same file or replacing a playing media object with another within the same scene specified by directory indices. Each variant may be called with or without an object mapping. Additionally, a JUMPTO command may replace a currently playing media object stream with a media object from the locally stored persistent object library 75.

While most of the interaction control functions can be handled by the client 20 using the rendering engine 74 in conjunction with the interaction manager 41, some control instances may need to be handled at a lower level and are passed back to the server 21.

This includes commands for non-linear navigation, such as jumping to hyperlinks and dynamic scene composition, with the exception of commands instructing insertion of objects from the object library 75.

The object library 75 of Figure 8 is a persistent, local media object library. Objects can be inserted into or removed from this library through special object control packets 68 known as object library control packets, and Scene Definition packets 66 which have the

ObjLibrary mode bit field set. The object library control packet defines the action to be performed with the object, including inserting, updating, purging and querying the object library. The input data switch/demux 32 may route compressed data packets 52 directly to the object library 75 if the appropriate object library action (for example insert or update) is defined. As shown in the block diagram of Figure 48, each object is stored in the object library data store 75g as a separate stream; the library does not support multiple interleaved objects since addressing is based on the library ID that is the stream number.

Hence the library may contain up to 200 separate user objects, and the object library may be referenced using a special scene number (for example 250). The library also supports up to 55 system objects, such as default buttons, checkboxes, forms, etc. The library supports garbage collection, such that an object may be set to expire after a certain time period, at which time the object is purged from the library. For eachobject/stream, the information contained in an object library control packet is stored by the client 20, containing additional information for the stream/object including the library id 75a, version information 75b, object persist information 75c, access restrictions 75d, unique object identifier 75e and other state information75f. The object stream additionally includes compressed object data 52. The object library 75 may be queried by the interaction management engine 41 of Figure 8, as directed by the object control component 40. This is performed by reading and comparing the object identifier values sequentially for all objects in the library 75 to find a match against the supplied search key. The library query results 75i are returned to the interaction management engine 41, to be processed or sent to the server 21. The object library manager 75h is responsible for managing all interaction with the object library.

*Server Software

The purpose of the server system 21 is to (i) create the correct data stream for the client to decode and render (ii) to transmit said data reliably to the client over a wireless channel including TDMA, FDMA or CDMA systems, and (iii) to process user interaction. The content of the data stream is a function of the dynamic media composition process 76 and non-sequential access requirements imposed by non-linear media navigation. Both the client 20 and server 21 are involved in the DMC process 76. The source data for the composite data stream may come from either a single source or from multiple sources. In the single source case, the source should contain all of the optional data components that may be required to composite the final data stream. Hence this source is likely to contain a library of different scenes, and multiple data streams for the various media objects that are to be used for composition. Since these media objects may be composited simultaneously into a single scene, advanced non-sequential access capabilities are provided on the part of the server 21 to select the appropriate data components from each media object stream in order to interleave them into the final composite data stream to send to the client 20. In the multiple source case, each of the different media objects to be used in the composition can have individual sources. Having the component objects for a scene in separate sources relieves the server 21 of the complex access requirements, since each source need only be sequentially accessed, although there are more sources to manage.

Both source cases are supported. For download and play functionality, it is preferable to deliver one file containing the packaged content, rather than multiple data files. For streaming play, it is preferable to keep the sources separate, since this permits much greater flexibility in the composition process and permits it to be tailored to specific user needs such as targeted user advertising. The separate source case also presents a reduced load on server equipment since all file accesses are sequential.

Figure 14 is a block diagram of the local server component of an interactive multimedia player playing locally stored files. As shown in Figure 14, standalone players need a local client system 20 and a local single source server system 23.

As shown in Figure 15, streaming players need a local client system 20 and a remote multi-source server 24. However, a player is also able to play local files and streaming content simultaneously, so the client system 20 is also able to simultaneously accept data from both a local server and a remote server. The local server 23 or the remote server 24 may constitute the server 21.

Referring to the simplest case with passive media playback in Figure 14, the local server 23 opens an object oriented data file 80 and sequentially reads its contents, passing the data 64 to the client 20. Upon a user command performed at user control 68, the file reading operation may be stopped, paused, continued from its current position, or restarted from the beginning of the object oriented data file 80. The server 23 performs two functions : accessing the object oriented data file 80, and controlling this access. These can be generalised into the multiplexer/data source manager 25 and the dynamic media composition engine 76.

In the more advanced case with local playback of video and dynamic media composition (Figure 14), it is not possible for the client to merely sequentially read one predetermined stream with multiplexed objects, because the contents of the multiplexed stream are not known when the object oriented data file 80 is created. Therefore, the local object oriented data file 80 includes multiple streams for each scene which are stored contiguously. The local server 23 randomly accesses each stream within a scene and selects the objects which need to be sent to the client 20 for rendering. In addition, a persistent object library 75 is maintained by the client 20 and can be managed from the remote server when online.

This is used to store commonly downloaded objects such as checkbox images for forms.

The data source manager/multiplexer 25 of Figure 14 randomly accesses the object oriented data file 80, reads data and control packets from the various streams in the file used to compose the display scene, and multiplexes these together to create the composite packet stream 64 that the client 20 uses to render the composite scene. A stream is purely conceptual as there is no packet indicating the start of a stream. There is, however, an end of stream packet to demarcate stream boundaries as shown at 53 in Figure 5. Typically, the first stream in a scene contains descriptions of the objects within the scene. Object control packets within the scene may change the source data for a particular object to a different stream. The server 23 then needs to read more than one stream simultaneously from within an object oriented data file 80 when performing local playback. Rather than creating separate threads, an array or linked list of streams can be created. The mutliplexer/data source manager 25 reads one packet from each stream in a round-robin fashion. At a minimum, each stream needs to store the current position in the file and a list of referencing objects.

In this case, the dynamic media composition engine 76 of Figure 14, upon the receipt of user control information 68 from the client 20, selects the correct combination of objects to be composited together, and ensures that themutliplexer/data source manager 25 knows where to find these objects, based on directory information provided to the dynamic media composition engine 76 by the multiplexer/data source manager 25. This may also require an object mapping function to map the storage object identifier with the run time object identifier, because they can differ depending upon the composition. A typical situation where this may occur is when multiple scenes in a file 80 may wish to share a particular video or audio object. Since a file may contain multiple scenes, this can be achieved by storing shared content in a special"library"scene. Objects within a scene have object IDs ranging from 0-200, and every time a new scene definition packet is encountered, the scene is reset with no objects. Each packet contains a base header that specifies the type of the packet as well as the object ID of the referenced object. An object ID of 254 represents the scene, whilst an object ID of 255 represents the file. When multiple scenes share an object data stream, it is not known what object IDs will have already been allocated for different scenes; hence, it is not possible to preselect the object IDs in the shared object stream, as these may already be allocated in a scene. One way to get around this problem is to have unique IDs within a file, but this increases storage space and makes it more difficult to manage sparse object IDs. The problem is solved by allowing each scene to use its own object IDs and when a packet from one scene indicates a jump to another scene, it specifies an object mapping between IDs from each scene.

When packets are read from the new scene, the mapping is used to convert the object IDs.

Object mapping information is expected to be in the same packet as a JUMPTO command.

If this information is not available, then the command is simply ignored. Object mappings may be represented using two arrays: one for the source object IDs which will be encountered in the stream, and the other for destination object IDs which the source object IDs will be converted to. If an object mapping is present in the current stream, then the destination object IDs of the new mapping are

converted using the object mapping arrays of the current stream. If an object mapping is not specified in the packet, then the new stream inherits the object mapping of the current stream (which may be null). All object IDs within a stream should be converted. For example, parameters such as: base header IDs, other IDs, button IDs, copyFrame IDs, and overlapping IDs should all be converted into the destination object IDs.

In the remote server scenario, shown in Figure 15, the server is remote from the client, so that data 64 will be streamed to the client. The media player client 20 is designed to decode packets received from the server 24 and to send back user operations 68 to the server. In this case, it is the remote server's 24 responsibility to respond to user operations (such as clicking an object), and to modify the packet stream 64 being sent to the client. In this case, each scene contains a single multiplexed stream (composed of one or more objects).

In this scenario, the server 24 composes scenes in real-time by multiplexing multiple object data streams based on client requests to construct a single multiplexed packet stream 64 (for any given scene) that is streamed to the client for playback. This architecture allows the media content being played back to change, based on user interaction. For example, two video objects may be playing simultaneously. When the user clicks or taps on one, it changes to a different video object, whilst the other video object remains unchanged. Each video may come from a different source, so the server opens both sources and interleaves the bit streams, adding appropriate control information and forwarding the new composite stream to the client. It is the server's responsibility to modify the stream appropriately before streaming it to the client.

Figure 15 is a block diagram of a remote streaming server 24. As shown, the remote server 24 has two main functional components similar to the local server: the data stream manager 26 and the dynamic media composition engine 76. However, the server intelligent multiplexer 27 can take input from multiple data stream manager 26 instances, each having a single data source and from the dynamic media composition engine 76, instead of from a single manager with multiple inputs. Along with the object data packets that are multiplexed together from the source (s), the intelligent multiplexer 27 inserts additional control packets into the packet stream to control the rendering of the component objects in the composite scene. The remote data stream managers 26 are also simpler, as they only perform sequential access. In addition to this, the remote server includes an XML parser 28 to enable programmable control of the dynamic media composition through an IAVML script 29. The remote server also accepts a number of inputs from the server operator database 19 to further control and customize the dynamic media composition process 76. Possible inputs include the time of day, day of the week, day of the year, geographic location of the client, and a user's demographic data, such as gender, age, any stored user profiles, etc. These inputs can be utilized in an IAVML script as variables in conditional expressions. The remote server 24 is also responsible for passing user interaction information such as object selections and form data back to the server operator's database 19 for later follow up processing such as data mining, etc.

As shown in Figure 15, the DMC engine 76 accepts three inputs and provides three outputs. The inputs include an XML based script, user input and database information.

The XML script is used to direct the operation of the DMC engine 76 by specifying how to compose the scene being streamed to the client 20. The composition is mediated by possible input from the user's interaction with objects in the current scene that have DMC control operations attached to them, or from input from a separate database. This database may contain information relating to time of day/date, the client's geographic location or the user's profile. The script can direct the dynamic composition process based on any combination of these inputs. This is performed by the DMC process by instructing the data stream managers to open a connection to and read the appropriate object data requried for the DMC operation, it also instructs the intelligent multiplexer to modify its interleaving of object packets received from the data stream managers and the DMC engine 76 to effect the removal, insertion or replacement of objects in a scene. The DMC engine 76 also optionally generates and attaches control information to objects according to the object control specifications for each in the script and provides this to the intelligent multiplexor for streaming to the client 20 as part of the object. Hence all of the processing is performed by the DMC engine 76 and no work is performed by the client 20 other than rendering the self-contained objects according to the parameters provided by any object control information. The DMC process 76 is capable of altering both objects in a scene and scenes in videos.

In contrast to this process is the process required to perform similar functionality in MPEG4. This does not use a scripting language but relies on the BIFS. Hence any modification of scenes requires the separate modification/insertion of the (i) BIFS, (ii) object descriptors, (iii) object shape information, and (iii) video object data packets. The BIFS has to be updated at the client device using a special BIFS-Command protocol. Since MPEG4 has separate but interdependent data components to define a scene, a change in composition cannot be achieved by simply multiplexing the object data packets (with or without control information) into a packet stream, but requires remote manipulation of the BIFS, multiplexing of the data packets and shape information, and the creation and transmision of new object descriptor packets. In addition, if advanced interactive functionality is required for MPEG4 objects, separately written Java programs are sent to the BIFS for execution by the client, which entails a significant processing overhead.

The operation of the local client performing Dynamic Media Composition (DMC) is described by the flow chart shown in Figure 16. In step s301, the Client DMC Process begins and immediately starts providing object compositing information to the data steam manager, facilitating multi-object video playback as shown in step s302. The DMC checks the user command list and the availability of further multimedia objects to ensure the video is still playing (step s303); if there is no more data or the user has stopped video playback, the Client DMC process ends (step s309). If, at step s303, video playback is to continue, the DMC process will browse the user command list and object control data for any initiated DMC actions. As shown in step s304, if no actions are initiated, the process returns to step s302 and video playback continues. However, if a DMC action DMC process sends instructions to the local data source manager to read the modified object stream from the local source, as shown in step s306; the process then returns to step s304 to check for further initiated DMC actions. If the target objects are stored remotely, the local DMC process sends appropriate DMC instuctions to the remote server, as shown in step s308. Alternativly, the DMC action may require target objects to be sourced both locally and remotely, as shown in step s307, thus appropriate DMC actions are executed by the local DMC process (step s306), and DMC instructions are sent to the remote server for processing (step s308). It is clear from this discussion that the local server supports hybrid, multi-object video playback, where source data is derived both locally and remotely.

The operation of the Dynamic Media Composition Engine 76 is described by the flow chart shown in Figure 17. The DMC process begins in step s401, and enters a wait state, step s402, until a DMC request is received. On receipt of a request the DMC engine 76 queries the request type at steps s403, s404 and s405. If at step s403 the request is determined to be an object Replace action, then two target objects exist: an active target object and a new target object to be added to the stream. First, the data stream manager is instructed, at step s406, to delete the active target object packets from the multiplexed bitstream, and to stop reading the active target object stream from storage. Subsequently, the datastream manager is instructed, at step s408, to read the new target object stream from storage, and to interleave these packets into the transmitted multiplex bit stream.

The DMC engine 76 then returns to its wait state at step s402. If at step s403 the request was not an object Replace action, then at step s404 if the action type is an object remove action, then one target object exists, which is an active target object. The object Remove action is processed at step s407, where the data stream manager is instructed to delete the active target object packets from the multiplex bitstream, and to stop reading the active target object stream from storage. The DMC engine 76 then returns to its wait state at step s402. If at step s404 the requested action was not an object Remove action, then at step s405 if the action is an object Add action, then one target object exists, which is a new target object. The object Add action is processed at step s408, where the datastream manager is instructed to read the new target object stream from storage, and to interleave these packets into the transmitted multiplex bit stream. The DMC engine 76 then returns to its wait state at step s402. Finally, if the requested DMC action is not an object Replace action (at step s403), or an object Remove action (at step s404), or an object Add action (at step s405), then the DMC engine 76 ignores the request and returns to its wait state at step s402.

Video Decoder

It is inefficient to store, transmit and manipulate raw video data, and so computer video systems normally encode video data into a compressed format. The section following this one describes how video data is encoded into an efficient, compressed form. This section describes the video decoder, which is responsible for generating video data from the compressed data stream. The video codec supports arbitrary-shaped video objects. It represents each video frame using three information components: a colour map, a tree based encoded bitmap, and a list of motion vectors. The colour map is a table of all of the colours used in the frame, specified in 24 bit precision with 8 bits allocated for each of the red, green and blue components. These colours are referenced by their index into the colour map. The bitmap is used to define a number of things including: the colour of pixels in the frame to be rendered on the display, the areas of the frame that are to be made transparent, and the areas of the frame that are to be unchanged. Each pixel in each encoded frame may be allocated to one of these functions. Which of these roles a pixel has is defined by its value. For example, if an 8 bit colour representation is used, then colour value OxFF may be assigned to indicate that the corresponding on screen pixel is not to be changed from its current value, and the colour value ofOxFE may be assigned to indicate that the corresponding on screen pixel for that object is to be transparent. The final colour of an on-screen pixel, where the encoded frame pixel colour value indicates it is transparent, depends on the background scene colour and any underlying video objects.

The specific encoding used for each of these components that makes up an encoded video frame is described below.

The colour table is encoded by first sending an integer value to the bit stream to indicate the number of table entries to follow. Each table entry to be sent is then encoded by first sending its index. Following this, a one bit flag is sent for each colour component (Rf, Gf andBf) indicating, if it is ON, that the colour component is being sent as a full byte, and if the flag is OFF that the high order nibble (4 bits) of the respective colour component will be sent and the low order nibble is set to zero. Hence the table entry is encoded in the following pattern where the number or C language expression in the parenthesis indicates the number of bits being sent: R(Rf ? 8 : 4), G (Gf? 8: 4), B (Bf? 8: 4).

The motion vectors are encoded as an array. First, the number of motion vectors in the array is sent as a 16 bit value, followed by the size of the macro blocks, and then the array of motion vectors. Each the entry in the array contains the location of the macro block and the motion vector for the block. The motion vector is encoded as two signed nibbles, one each for the horizontal and vertical components of the vector.

The actual video frame data is encoded using a preordered tree traversal method. There are two types of leaves in the tree: transparent leaves, and region colour leaves. The transparent leaves indicate that the onscreen displayed region indicated by the leaf will not be altered, while the colour leaves will force the onscreen region to the colour specified by the leaf. In terms of the three functions that can be assigned to any encoded pixel as previously described, the transparent leaves would correspond to the colour value of OxFF while pixels with a value ofOxFE indicating that the on screen region is to be forced to be transparent are treated as normal region colour leaves. The encoder starts at the top of the tree and for each node stores a single bit to indicate if the node is a leaf or a parent. If it is a leaf, the value of this bit is set to ON, and another single bit is sent to indicate if the region is transparent (OFF), otherwise it is set to ON followed by a another one bit flag to indicate if the colour of the leaf is sent as an index into a FIFO buffer or as the actual index into the colour map. If this flag is set to OFF, then a two bit codeword is sent as the index of one of the FIFO buffer entries. If the flag is ON, this indicates that the leaf colour is not found in the FIFO, and the actual colour value is sent and also inserted into the FIFO, pushing out one of the existing entries. If the tree node was a parent node, then a single OFF bit is stored, and each of the four child nodes are then individually stored using the same method. When the encoder reaches the lowest level in the tree, then all nodes are leaf nodes and the leaf/parent indication bit is not used, instead storing first the transparency bit followed by the colour codeword. The pattern of bits sent can be represented as shown below. The following symbols are used: node type (N), transparent (T), FIFO Predicted colour (P), colour value (C), FIFO index (F)
EMI64.1

Figure 49 is a flowchart showing the principal steps of one embodiment of the video frame decoding process. The video frame decoding process begins at step s2201 with a compressed bit stream. A layer identifier, which is used to physically separate the various information components within the compressed bit stream, is read from the bit stream at step s2202. If the layer identifier indicates the start of the motion vector data layer, step s2203 proceeds to step s2204 to read and decode the motion vectors from the bit stream and perform the motion compensation. The motion vectors are used to copy the indicated macro blocks from the previously buffered frame to the new locations indicated by the vectors. When the motion compensation process is complete, the next layer identifier is read from the bit stream at step s2202. If the layer identifier indicates the start of the quad tree data layer, step s2205 proceeds to step s2206, and initialises the FIFO buffer used by the read leaf colour process. Next, the depth of the quad tree is read from the compressed bit stream at step s2207, and is used to initialize the quad tree quadrant size. The compressed bitmap quad tree data is now decoded at step s2208. As the quad tree data is decoded, the region values in the frame are modified according to the leaf values. They may be overwritten with new colours, set to transparent, or left unchanged.

When the quad tree data is decoded, the decode process reads the next layer identifier from the compressed bit stream at step s2202. If the layer indicates the start of the colour map data layer, step s2209 proceeds to step s2210 which reads the number of colours to be updated from the compressed bit stream. If there are one or more colours to update at steps2211, the first colour map index value is read from the compressed bit stream at step s2212, and the colour component values are read from the compressed bit stream at step s2213. Each colour update is in turn read through stepss2211, s2212, and s2213 until all of the colour updates have been performed, at which time step s2211 proceeds to step s2202 to read a new layer identifier from the compressed bit stream. If the layer identifier is an end of data indentifier, step s2214 proceeds to step s2215 and ends the video frame decoding process. If the layer identifier is unknown through steps s2203, s2205, s2209, and s2214, the layer identifier is ignored, and the process returns to step s2202 to read the next layer identifier.

Figure 50 is a flowchart showing the principal steps of one embodiment of a quad tree decoder with bottom-level node type elimination. This flowchart implements a recursive method, calling itself recursively for each tree quadrant processed. The quad tree decoding process begins at step s2301, having some mechanism of recognising the depth and position of the quadrant to be decoded. If at step s2302 the quadrant is a non-bottom quadrant, the node type is read from the compressed bit stream at step s2307. If the node type is a parent node at step s2308, then four recursive calls are in turn made to the quad tree decoding process for the top left quadrant at step s2309, the top right quadrant and steps2310, the bottom left quadrant at steps2311, the bottom right quadrant at steps2312 ; subsequently this iteration of the decoding process ends at step s2317. The particular order in which the recursive calls are made for each quadrant is arbitrary, however the order is the same as the quad tree decomposition process performed by the encoder. If the node type is a leaf node, the process continues from step s2308 to s2313, and the leaf type value is read from the compressed bit stream. If the leaf type value indicates a transparent leaf at step s2314, the decoding process ends at step s2317. If the leaf is not transparent, the leaf colour is read from the compressed bit stream at step s2315. The leaf read colour value function employs a FIFO buffer, described herein. Subsequently at step s2316 the image quadrant is set to the appropriate leaf colour value; this may be the background object colour or the leaf colour as indicated. After the image update is complete, the quad tree decode function ends this iteration at step s2317. The recursive calls to the quad tree decode function continue until a bottom level quadrant is reached. At this level there is no need to include in the compressed bit stream a parent/leaf node indicator, as each node at this level is aleaf ; hence step s2302 proceeds to step s2303 and reads immediately the leaf type value. If the leaf is not transparent at step s2304, then the leaf colour value is read from the compressed bit stream at step s2305, and the image quadrant colours are updated appropriately at step s2306. This iteration of the decoding process ends at step s2317.

The recursive process executions of the quad tree decoding process continue until all leaf nodes in the compressed bit stream have been decoded.

Figure 51 shows the steps executed in reading a quad tree leaf colour, beginning at step s2401. A single flag is read from the compressed bit stream at step s2402. This flag indicates if the leaf colour is to be read from the FIFO buffer or directly from the bit

stream. If, at step s2403, the leaf colour is not to be read from the FIFO, the leaf colour value is read from the compressed bit stream at step s2404, and is stored in the FIFO buffer at step s2405. Storing the newly read colour in the FIFO pushes out the least recently added colour in the FIFO. The read leaf colour function ends at step s2408, after updating the FIFO. If however the leaf colour is already stored in the FIFO, the FIFO index codeword is read from the compressed bit stream at steps2406. The leaf colour is then determined, at step s2407, by indexing into the FIFO, based on the recently read codeword. The read leaf colour process ends at step s2408.

Video Encoder

To this point, the discussion hasfocussed on the manipulation of pre-existing video objects and files which contain video data. The previous section described how compressed video data is decoded to produce raw video data. In this section, the process of generating this data is discussed. The system is designed to support a number of different codecs. Two such codecs are described here; others that may also be used include the
MPEG family and H. 261 and H. 263 and their successors.

The encoder comprises ten main components, as shown in Figure 18. The components can be implemented in software, but to enhance the speed of the encoder, all the components can be implemented in an application-specific integrated circuit (ASIC) developed specifically to execute the steps of the encoding process. An audio coding component 12 compresses input audio data. The audio coding component 12 may use adaptive delta pulse code modulation (ADPCM) according to either ITU specification
G. 723 or the IMA ADPCM codec. A scene/object control data component 14 encodes scene animation and presentation parameters associated with the input audio and video which determine the relationships and behaviour of each input video object. An input colour processing component 10 receives and processes individual input video frames and eliminates redundant and unwanted colours. This also removes unwanted noise from video images. Optionally, motion compensation is performed on the output of the input colour processor 10 using the previously encoded frame as a basis. A colour difference management and synchronisation component 16 receives the output of the input colour processor 10, and determines the encoding using the optionally motion-compensated, previously encoded frame as a basis. The output is then provided to both a combined spatial/temporal coder 18 to compress the video data, and to a decoder 20 which executes the inverse function to provide the frame to the motion compensation component 11 after a one frame delay 24. A transmission buffer 22 receives the output of the spatial/temporal coder 18, the audio coder 12 and the control data component 14. The transmission buffer 22 manages transmission from a video server housing the encoder, by interleaving encoded data and controlling data rates via feedback of rate information to the combined spatial/temporal coder 18. If required, the encoded data can be encrypted by an encryption component 28 for transmission.

The flow chart of Figure 19 describes the main steps executed by the encoder. The video compression process begins at step s501, entering a frame compression loop (s502 to s521), and ending at step s522 when, at step s502, there are no video data frames remaining in the input video data stream. The raw video frame is fetched from the input data stream in step s503. At this point, it may be desired to perform spatial filtering.

Spatial filtering is performed to lower the bit rate or total bits of the video being generated, but spatial filtering also lowers the fidelity. If it is determined by step s504 that spatial filtering is to be performed, a colour difference frame is calculated at step s505 between the current input video frame and the previously processed or reconstructed video frame.

It is preferable to perform the spatial filtering where there is movement, and the step of calculating the frame difference indicates where there is movement; if there is no difference, then there is no movement, and a difference in regions of a frame indicates movement for those regions. Subsequently, localised spatial filtering is performed on the input video frame at step s506. This filtering is localised such that only image regions that have changed between frames are filtered. If desired, the spatial filtering may also be performed on I frames. This can be carried out using any desired technique including inverse gradient filtering, median filtering, and/or a combination of these two types of filtering, for example. If it is desired to perform spatial filtering on a key frame and also to calculate the frame difference in stepS505, the reference frame used to calculate the difference frame may be an empty frame.

Colour quantisation is performed at step s507 to remove statistically insignificant colours from the image. The general process of colour quantisation is known with respect to still images. Example types of colour quantisation which may be utilised by the invention include, but are not limited to, all techniques described in and referenced by U. S Patent
Nos. 5,432,893 and 4,654,720 which are incorporated by reference. Also incorporated by reference are all documents cited by and referenced in these patents. Further information about the colour quantisation step s507 is explained with reference to elements10a, IOb, and10c of Figure 20. If a colour map update is to be performed for this frame, flow proceeds from step s508 to step s509. In order to achieve the highest quality image, the colourmap may be updated every frame. However, this may result in too much information being transmitted, or may require too much processing. Therefore, instead of updating the colourmap every frame, the colour map may be updated every n frames, where n is an integer equal to or greater than 2, preferably less than 100, and more preferably less than 20. Alternatively, the colour map may be updated every n frames on average, where n is not required to be an integer, but may be any value including fractions greater than 1 and less than a predetermined number, such as 100 and more preferably less than 20. These numbers are merely exemplary and, if desired, the colour map may be updated as often or as infrequently as desired.

When there is a desire to update the colour map, step s509 is performed in which a new colour map is selected and correlated with the previous frame's colour map. When the colour map changes or is updated, it is desirable to keep the colour map for the current frame similar to the colour map of the previous frame so that there is not a visible discontinuity between frames which use different colour maps.

If at step s508 no colour map is pending (e. g. there is no need to update the colour map), the previous frame's colour map is selected or utilised for this frame. At steps510, the quantised input image colours are remapped to new colours based on the selected colour map. Step s510 corresponds to block10d of Figure 20. Next, frame buffer swapping is performed in steps511. Frame buffer swapping at step s511 facilitates faster and more memory efficient encoding. As an exemplary implementation of frame buffer swapping, two frame buffers may be used. When a frame has been processed, the buffer for this frame is designated as holding a past frame, and a new frame received in the other buffer is designated as being the current frame. This swapping of frame buffers allows an efficient allocation of memory.

A key reference frame, also referred to as a reference frame or a key frame, may serve as a reference. If step s512 determines that this frame (the current frame) is to be encoded as, or is designated as, a key frame, the video compression process proceeds directly to step s519 to encode and transmit the frame. A video frame may be encoded as a key frame for a number of reasons, including: (i) it is the first frame in a sequence of video frames following a video definition packet, (ii) the encoder detects a visual scene change in the video content, or (iii) the user has selected key frames to be inserted into the video packet stream. If the frame is not a key frame, the video compression process calculates, at step s513, a difference frame between the current colour map indexed frame and the previous reconstructed colour map indexed frame. The difference frame, the previous reconstructed colour map indexed frame, and the current colour map indexed frame are used at step s514 to generate motion vectors, which are in turn used to rearrange the previous frame at steps515.

The rearranged previous frame and the current frame are now compared at step s516 to produce a conditional replenishment image If blue screen transparency is enabled at steps517, step s518 will drop out regions of the difference frame that fall within the blue screen threshold. The difference frame is now encoded and transmitted at steps519. Step s519 is explained in further detail below with reference to Figure 24. Bit rate control parameters are established at step s520, based on the size of the encoded bit stream.

Finally the encoded frame is reconstructed at step s521 for use in encoding the next video frame, beginning at step s502.

The input colour processing component 10 of Figure 18 performs reduction of statistically insignificant colours. The colour space chosen to perform this colour reduction is unimportant as the same outcome can be achieved using any one of a number of different colour

spaces.

The reduction of statistically insignificant colours may be implemented using various vector quantisation techniques as discussed above, and may also be implemented using any other desired technique including popularity, median cut, k-nearest neighbour and variance methods as described in S. J. Wan, P. Prusinkiewicz, S. K. M. Wong,"Variance Based Color Image Quantization for Frame Buffer Display.", Color Research and Application, Vol. 15, No.1, Feb 1990, which is incorporated by reference. As shown in Figure 20, these methods may utilise an initial uniform or non-adaptive quantisation step IOa to improve the performance of the vector quantisation algorithm IOb by reducing the size of the vector space. The choice of method is made to maintain the highest amount of time correlation between the quantised video frames, if desired. The input to this process is the candidate video frame, and the process proceeds by analysing the statistical distribution of colours in the frame. In 10c, the colours which are used to represent the image are selected. With the technology available now for some hand-held processing devices or personal digital assistants, there may be a limit of simultaneously displaying 256 colours, for example. Thus,10c may be utilised to select 256 different colours to be used to represent the image. The output of the vector quantisation process is a table of representative colours for the entire frame I Oc that can be limited in size. In the case of the popularity methods, the most frequent N colours are selected. Finally, each of the colours in the original frame is remapped 1 Od to one of the colours in the representative set.

The colour management components IOb,10c and10d of the Input Colour Processing component 10 manages the colour changes in the video. The input colour processing component 10 produces a table containing a set of displayed colours. This set of colours changes dynamically over time, given that the process is adaptive on a per frame basis.

This permits the colour composition of the video frames to change without reducing the image quality. Selecting an appropriate scheme to manage the adaptation of the colour map is important. Three distinct possibilities exist for the colour map: it may be static, segmented and partially static, or fully dynamic. With a fixed or static colour map, the local image quality will be reduced, but high correlation is preserved from frame to frame, leading to high compression gains. In order to maintain high quality images for video where scene changes may be frequent, the colour map should be able to adapt instantaneously. Selecting a new optimal colour map for each frame has a high bandwidth requirement, since not only is the colour map updated every frame, but also a large number of pixels in the image would need to be remapped each time. This remapping also introduces the problem of colour map flashing. A compromise is to only permit limited colour variations between successive frames. This can be achieved by partitioning the colour map into static and dynamic sections, or by limiting the number of colours that are allowed to vary per frame. In the first case, the entries in the dynamic section of the table can be modified, which ensures that certain predefined colours will always be available. In the other scheme, there are no reserved colours and any may be modified. While this approach helps to preserve some data correlation, the colour map may not be able to adapt quickly enough in some cases to eliminate image quality degradation. Existing approaches compromise image quality to preserve frame-to-frame image correlation.

For any of these dynamic colour map schemes, synchronisation is important to preserve temporal correlations. This synchronisation process has three components: 1. Ensuring that colours carried over from each frame into the next are mapped to the same indices over time. This involves resorting each new colour map in relation to the current one.

2. A replacement scheme is used for updating the changed colour map. To reduce the amount of colour flashing, the most appropriate scheme is to replace the obsolete colour with the most similar new replacement colour.

3. Finally, all existing references in the image to any colour that is no longer supported are replaced by references to currently supported colours.

Following the input colour processing 10 of Figure 18, the next component of the video encoder takes the indexed colour frames and optionally performs motion compensation 11.

If motion compensation is not performed, then the previous frame from the frame buffer 24 is not modified by the motion compensation component 11 and is passed directly to the colour difference management and synchronisation component 16. The preferred motion compensation method starts by segmenting the video frame into small blocks and determining all blocks in a video frame where the number of pixels needing to be replenished or updated and are not transparent exceeds some threshold. The motion compensation process is then performed on the resultant pixel blocks. First, a search is made in the neighbourhood of the region to determine if the region has been displaced from the previous frame. The traditional method for performing this is to calculate the mean square error (MSE) or sum square error (SSE) metric between the reference region and a candidate displacement region. As shown in Figure 22, this pr performed using an exhaustive search or one of a number of other existing search techniques, such as the 2D logarithmic11 a, three step llb or simplified conjugate direction search llc. The aim of this search is to find the displacement vector for the region, often called the motion vector. Traditional metrics do not work with indexed/colour mapped image representations because they rely on the continuity and spatio-temporal correlation that continuous image representations provide. With indexed representations, there is very little spatial correlation and no gradual or continuous change of pixel colour from frame to frame; rather, changes are discontinuous as the colour index jumps to new colour map entries to reflect pixel colour changes. Hence a single index/pixel changing colour will introduce large changes to theMSE or SSE, reducing the reliability of these metrics. Hence a better metric for locating region displacement is where the number of pixels that are different in the previous frame compared to the current frame region is the least if the region is not transparent. Once the motion vector is found, the region is motion-compensated by predicting the value of the pixels in the region from their original location in the previous frame according to the motion vector. The motion vector may be zero if the vector giving the least difference corresponds to no displacement. The motion vector for each displaced block, together with the relative address of the block, is encoded into the output bitstream. Following this, the colour difference management component 16 calculates the perceptual difference between the motion-compensated previous frame and the current frame.

The colour difference management component 16 is responsible for calculating the perceived colour difference at each pixel between the current and preceding frame. This perceived colour difference is based on a similar calculation to that described for the perceptual colour reduction. Pixels are updated if their colour has changed more than a given amount. The colour difference management component 16 is also responsible for purging all invalid colour map references in the image, and replacing these with valid references, generating a conditional replenishment image. Invalid colour map references may occur when newer colours displace old colours in the colour map. This information is then passed to the spatial/temporal coding component 18 in the video encoding process.

This information indicates which regions in the frame are fully transparent, and which need to be replenished, and which colours in the colour map need to be updated. All regions in a frame not being updated are identified by setting the value of the pixel to a predetermined value that has been selected to represent non update. The inclusion of this value permits the creation of arbitrarily shaped video objects. To ensure that prediction errors do not accumulate and degrade the image quality, a loop filter is used. This forces the frame replenishment data to be determined from the present frame and the accumulated previous transmitted data (the current state of the decoded image), rather than from the present and previous frames. Figure 21 provides a more detailed view of the colour difference management component 16. The current frame store 16a contains the resultant image from the input colour processing component 10. The previous frame store 16b contains the frame buffered by the 1 frame delay component 24, which may or may not have been motion-compensated by the motion compensation component 11. The colour difference management component 16 is portioned into two main components: the calculation of perceived colour differences between pixels 16c, and cleaning up invalid colour map references 16f. The perceived colour differences are evaluated with respect to a threshold 16d to determine which pixels need to be updated, and the resultant pixels are optionally filtered 16e to reduce the data rate. The final update image is formed 16g from the output of the spatial filter 16e and the invalid colour map references 16f and is sent to the spatial encoder 18.

This results in a conditional replenishment frame which is now encoded. The spatial encoder 18 uses a tree splitting method to recursively partition each frame into smaller polygons according to a splitting criteria. A quad tree split 23d method used, as is shown in Figure 23. In one instance, that of zeroth order interpolation, this attempts to represent the image 23a by a uniform block, the value of which is equal to the global mean value of the image. In another instance, first or second order interpolation may be used. If, at some locations of the image, the difference between this representative value and the real value exceeds some tolerance threshold, then the block is recursively subdivided uniformly, into two or four subregions, and a new mean is calculated for each subregion. For lossless image encoding, there is no tolerance threshold. The tree structures 23d, 23e, 23f are composed of nodes and pointers, where each node represents a region and contains pointers to any child nodes representing subregions which may exist. There are two types of nodes: leaf 23b and non-leaf 23c nodes. Leaf nodes 23b are those that are not further decomposed and as such have no children, instead containing a representative value for the implied region. Non-leaf nodes 23c do not contain a representative value, since these consist of further subregions and as such contain pointers to the respective child nodes.

These can also be referred to as parent nodes.

Dynamic Bitmap (Colour) Encoding
The actual encoded representation of a single video frame includes bitmap, colour map, motion vector and video enhancement data. As shown in Figure 24, the video frame encoding process begins at step s601. If (s602) motion vectors were generated via the motion compensation process, then the motion vectors are encoded at step s603. If (s604) the colour map has changed since the previous video frame, the new colour map entries are encoded at step s605. The tree structure is created from the bitmap frame at step s606 and is encoded at step s607. If (s608) video enhancement data is to be encoded, the enhancement data is encoded at step s609. Finally, the video frame encoding process ends at steps610.

The actual quadtree video frame data is encoded using a preordered tree traversal method.

There may be two types of leaves in the tree: transparent leaves and region colour leaves.

The transparent leaves indicate that the region indicated by the leaf is unchanged from its previous value (these are not present in video key frames), and the colour leaves contain the region colour. Figure 26 represents a pre-ordered tree traversal encoding method for normal predicted video frames with zeroth order interpolation and bottom level node type elimination. The encoder of Figure 26 begins at step s801, initially adding a quad tree layer identifier to the encoded bit stream at step s802. Beginning at the top of the tree, step s803, the encoder gets the initial node. If, at step s804, the node is a parent node, the encoder adds a parent node flag (a single ZERO bit) to the bit stream at step s805.

Subsequently, the next node is fetched from the tree at step s806, and the encoding process returns to step s804 to encode subsequent nodes in the tree. If at step s804 the node is not a parent node, i. e., it is a leaf node, the encoder checks the node level in the tree at step s807. If at step s807 the node is not at the bottom of the tree, the encoder adds a leaf node flag (a single ONE bit) to the bit stream at step s808. If the leaf node region is transparent at step s809, a transparent leaf flag (a single ZERO bit) is added to the bit stream at step s810 ; otherwise, an opaque leaf flag (single ONE bit) is added to the bit stream at step s811. The opaque leaf colour is then encoded at steps812, as shown in Figure 27. If, however, at step s807 the leaf node is at the bottom level of the tree, then bottom level node type elimination occurs because all nodes are leaf nodes and the leaf/parent indication bit is not used, such that at step s813 four flags are added to the bit stream to indicate if each of the four leaves at this level are transparent (ZERO) or opaque (ONE).

Subsequently, if the top left leaf is opaque at steps814, then at step s815 the top left leaf colour is encoded as shown in Figure 27. Each of steps s814 and s815 are repeated for each leaf node at this second bottom level, as shown in steps s816 and s817 for the top right node, steps s818 and s819 for the bottom left node, and steps s820 and s821 for bottom right node. After the leaf nodes are encoded (from stepss810, s812, s820 or s821) the encoder checks whether further nodes remain in the tree at step s822. If no nodes remain in the tree, then the encoding process ends at step s823. Otherwise, the encoding process continues at step s806, where the next node is selected from the tree and the entire process restarts for the new node from step s804.

In the special case of video key frames (these are not predicted), these do not have transparent leaves and a slightly different encoding method is used, as shown in Figure 28.

The key frame encoding process begins at step s1001, initially adding a quad tree layer identifier to the encoded bit stream at steps1002. Beginning at the top of the tree, steps1003, the encoder gets the initial node. If, at steps1004, the node is a parent node, the encoder adds a parent node flag (a single ZERO bit) to the bit stream at steps1005 ; subsequently, the next node is fetched from the tree at steps1006, and the encoding process returns to steps1004 to encode subsequent nodes in the tree. If however at steps1004 the node is not a parent node, i. e. it is a leaf node, the encoder checks the node level in the tree at stepsl007. If at step sl007 the node is greater than one level from the bottom of the tree the encoder adds a leaf node flag (a single ONE bit) to the bit stream at stepsl008. The opaque leaf colour is then encoded at steps1009, as shown in Figure 27.

If, however at stepsix07 the leaf node is one level from the bottom of the tree, then bottom level node type elimination occurs because all nodes are leaf nodes and the leaf/parent indication bit is not used. Thus at step s1010 the top left leaf colour is encoded as shown in Figure 27. Subsequently, at stepss 1 O 11, s 1012 ands1013, the opaque leaf colours are encoded similarly for the top right leaf, bottom left leaf and the bottom right leaf respectively. After the leaf nodes are encoded (from stepssix09 ors1013) the encoder checks whether further nodes remain in the tree at steps1014. If no nodes remain in the tree, then the encoding process ends at steps1015. Otherwise, the encoding process continues, at steps1006, where the next node is selected from the tree and the entire process restarts for the new node from steps1004.

The opaque leaf colours are encoded using a FIFO buffer as shown in Figure 27. The leaf colour encoding process begins at step s901. The colour to be encoded is compared with the four colours already in the FIFO, if at step s902 it is determined that the colour is in the FIFO buffer, then a single FIFO lookup flag (single ONE bit) is added to the bit stream at step s903, followed by, at step s904, a two bit codeword representing the colour of the leaf as an index into the FIFO buffer. This codeword indexes one of four entries in the FIFO buffer. For example, index values of 00,01 and 10 specify that the leaf colour is the same as the previous leaf, the previous different leaf colour before that, and the previous one before that respectively. If however at step s902 the colour to be encoded is not available in the FIFO, a send colour flag (a single ZERO bit) is added to the bit stream at step s906, followed by N bits, at step s906, representing the actual colour value.

Additionally, the colour is added to the FIFO, pushing out one of the existing entries. The colour leaf encoding process ends then at step s907.

The colourmap is similarly compressed. The standard representation is to send each index followed by 24 bits, 8 to specify the red component value, 8 for the green component and 8 for the blue. In the compressed format, a single bit flag indicates if each colour component is specified as a full 8-bit value, or just as the top nibble with the bottom 4 bits set to zero. Following this flag, the component value is sent as 8 or 4 bits depending on the flag. The flowchart of Figure 25 depicts one embodiment of a colour map encoding method using 8-bit colour map indices. In this implementation, the single bit flags specifying the resolution of the colour component for all the components of one colour are encoded prior to the colour components themselves. The colour map update process begins at step s701. Initially, a colour map layer identifier is added to the bit stream at step s702, followed by, at step s703, a codeword indicating the number of colour updates following. At step s704 the process checks a colour update list for additional updates; if no further colour updates require encoding, the process ends at steps717. If, however, colours remain to be encoded, then at step s705 the colour table index to be updated is added to the bit stream. For each colour there are typically a number of components (red, green and blue, for example), thus step s706 forms a loop condition around steps s707, s708, s709 ands710, processing each component separately. Each component is read from the data buffer at step s707. Subsequently, if, at step s708, the component low order nibble is zero, an off flag

(a single ZERO bit) is added to the bit stream at step s709, or if the low order nibble is non-zero, an on flag (a single ONE bit) is added to the bit stream at step s710. The process is repeated by returning to step s706, until no colour components remain. Subsequently, the first component is again read from the data buffer at steps711.

Similarly, step s712 forms a loop condition around stepss713, s714, s715 ands716, processing each component separately. Subsequently, if, at steps712, the component's low order nibble is zero, the component's high order nibble is added to the bit stream at steps713. Alternatively, if the low order nibble is non-zero, the component's 8-bit colour component is added to the bit stream at steps714. If further colour components remain to be added at steps715, the next colour component is read from the input data stream at steps716, and the process returns to step s712 to process this component. Otherwise, if no components remain at steps715, the colour map encoding process returns to step s704 to process any remaining colour map updates.

Alternate Encoding Method
In the alternate encoding method, the process is very similar to the first as shown in Figure 29 except that the input colour processing component 10 of Figure 18 does not perform colour reduction, but instead ensures that the input colour space is in YCbCr format, converting from RGB if required. There is no colour quantisation or colour map management required, thus steps s507 through s510 of Figure 19 are replaced by a single colour space conversion step, ensuring the frame is represented in YCbCr colour space.

The motion compensation component 11 of Figure 18 performs"traditional"motion compensation on the Y component and stores the motion vectors. The conditional replenishment images are then generated from the inter-frame coding process for each of the Y, Cb and Cr components using the motion vectors from the Y component. The three resultant difference images are then compressed independently after down-sampling the
Cb and Cr bitmaps by a factor of two in each direction. The bitmap encoding uses a similar recursive tree decomposition, but this time for each leaf that is not at the bottom of the tree, three values are stored: the mean bitmap value for the area represented by the leaf, and the gradients for the horizontal and verticaldirections. The flowchart of Figure 29 depicts the alternate bitmap encoding process, beginning at step sl 101. At stepsl 102 the image component (Y, Cb or Cr) is selected for encoding, then at stepsol03 the initial tree node is selected. If this node, at steps1104, is a parent node, a parent node flag(1 bit) is added to the bitstream. The next node is then selected from the tree at steps1106, and the alternate bitmap encoding process returns to stepsl 104. If at step sl 104 the new node is not at parent node, at steps 1107 the nodes depth in the tree is determined. If, at steps1107, the node is not at the bottom level of the tree, the node is encoded using the nonbottom leaf node encode method, such that at steps 1108 a leaf node flag(1 bit) is added to the bitstream. Subsequently if at steps1109 the leaf is transparent, a transparent leaf flag(1 bit) is added to the bitstream. If however the leaf is not transparent, an opaque leaf flag(1 bit) is added to the bitstream, subsequently at stepsl 112 the leaf colour mean value is encoded. The mean is encoded using a FIFO as in the first method by sending a flag and either the FIFO index in 2 bits or the mean itself in 8 bits. If at stepsl 113, the region is not an invisible background region (for use in arbitrary shaped video objects) then the leaf horizontal and vertical gradients are encoded at stepsu 114. Invisible background regions are encoded using a special value for the mean, for example OxFF. The gradients are sent as a 4 bit quantised value. If, however, at stepsl 107 it is determined that the leaf node is on the bottom most level of the tree, then the corresponding leaves are encoded as in the previous method by sending the bitmap value and noparent/lead indication flag.

Transparent and colour leaves are encoded as before using single bit flags. In the case of arbitrarily-shaped video, the invisible background regions are encoded by using a special value for the mean, for example OxFF, and in this case the gradient values are not sent
Specifically then at steps 1115 four flags are added to the bit stream to indicate if each of the four leaves at this level are transparent or opaque. Subsequently, if the top left leaf is opaque at steps 1116, then at steps 1117 the top left leaf colour is encoded as described above for opaque leaf colour encoding. Each of stepss1116 ands1117 are repeated for each leaf node at this bottom level, as shown in stepss1118 ands1119 for the top right node, stepssol20 andsol21 for the bottom left node, and stepssol22 andsol23 for the bottom right node. At the completion of leaf node encoding, the encoding process checks the tree for additional nodes at step s1124, ending at steps1125 if no nodes remain.

Alternatively, the next node is fetched at step s1106, and the process restarts at step sl 104.

The reconstruction in this case involves interpolating the values within each region identified by the leaves using first, second or third order interpolation and then combining the values for each of the Y, Cb and Cr components to regenerate the 24 bit RGB values for each pixel. For devices with 8 bit, colour mapped displays, quantisation of the colour is executed before display.

Encoding of Colour Prequantisation Data
For improved image quality, a first or second order interpolated coding can be used, as in the alternate encoding method previously described. In this case, not only was the mean colour for the region represented by each leaf stored, but also colour gradient information at each leaf. Reconstruction is then performed using quadratic or cubic interpolation to regenerate a continuous tone image. This may create a problem when displaying continuous colour images on devices with indexed colour displays. In these situations, the need to quantise the output down to 8 bits and index it in real time is prohibitive. As shown in Figure 47, in this case the encoder 50 can perform vector quantisation 02b of 24bit colour data 02a, generating colour pre-quantisation data. Colour quantisation information can be encoded using octree compression 02c, as described below. This compressed colour pre-quantisation data is sent with the encoded continuous tone image to enable the video decoder/player 38 to perform real-time colour quantisation 02d by applying the pre-calculated colour quantisation data, thus producing optionally 8-bit indexed colour video representation 02e in real-time. This technique can also be used when reconstruction filtering is used that generates a 24-bit result that is to be displayed on 8-bit devices. This problem can be resolved by sending a small amount of information to the video decoder 38 that describes the mapping from the 24 bit colour result to the 8 bit colour table. This process is depicted in the flowchart beginning with step s1201 in Figure 30, and includes the main steps involved in the pre-quantisation process to perform realtime colour quantisation at the client. All frames in the video are processed sequentially as indicated by the conditional block at stepsl202. If no frames remain, then the prequantisation process ends at steps1210. Otherwise at steps1203 the next video frame is fetched from the input video stream, and then at steps1204 vector pre-quantisation data is encoded. Subsequently, the non-index based colour video frames are encoded/compressed at stepsl205. The compressed/encoded frame data is sent to the client at steps1206, which the client subsequently decodes into a full-colour video frame at steps1207. The vector pre-quantisation data is now used for vector post-quantisation at steps1208, and finally the client renders the video frame at steps 1209. The process returns to steps 1202 to process subsequent video frames in the stream. The vector pre-quantisation data includes a three-dimensional array of size 32x64x32, where the cells in the array contain the index values for each r, g, b coordinate. Clearly, storing and sending a total of 32x64x32 = 65,536 index values is a large overhead that makes the technique impractical. The solution is to encode this information in a compact representation. One method, as shown in the flow chart of Figure 30 beginning at steps1301, is to encode this three dimensional array of indexes using an octree representation. The encoder 50 of Figure 47 may use this method. At steps1302, the 3D data set/video frame is read from the input source, such that Fj (r, g, b) represents all unique colours in the RGB colour space for all j pixels in the video frame. Subsequently at steps1303 N codebook vectorsVi are selected to best represent the 3D data set Fj (r, g, b). A three-dimensional array t [O..Rmax, I 0. Gm" .. Bma] is created in steps1304. For all cells in array t, the closest codebook vector Vi is determined in steps1305, and in steps1306 the closest codebook vector for each cell is stored in array t. If, at steps1307, previous video frames have been encoded such that a previous data array t exists, then step 1308 determines the differences between the current and previous t arrays ; subsequently, at steps1309, an update array is generated. Then, either the update array of steps1309 or the full array t is encoded at steps1310 using a lossy octree method.

This method takes the 3D array (cube) and recursively splits it in a similar manner to the quadtree based representation. Since the vector codebook(V ;)/colour map is free to change dynamically, this mapping information is also updated to reflect the changes in the colour map from frame to frame. A similar conditional replenishment method is proposed to perform this using the index value 255 to represent an unchanged coordinate mapping and other values to represent update values for the 3D mapping array. Like the spatial encoder, the process uses a preordered octree tree traversal method to encode the colour space mapping into the colour table. Transparent leaves indicate that the region of the colour space indicated by the leaf is unchanged and index leaves contain the colour

table index for the colour specified by the coordinates of the cell. The octree encoder starts at the top of the tree and for each node stores a single ONE bit if the node is a leaf, or a
ZERO bit if it is a parent. If it is a leaf and the colour space area is unchanged then another single ZERO bit is stored otherwise the corresponding colour map index is explicitly encoded as a n bit codeword. If the node was a parent node and a ZERO bit was stored, then each of the eight child nodes are recursively stored as described. When the encoder reaches the lowest level in the tree, then all nodes are leaf nodes and the leaf/parent indication bit is not used, instead storing first the unchanged bit followed by the colour index codeword. Finally, at steps1311, the encoded octree is sent to the decoder for post quantising data and at steps1312 the codebook vectorsV ;/colour map are sent to the decoder, thus ending the vector pre-quantisation process at stepsu313. The decoder performs the reverse process, vector post-quantisation, as shown in the flowchart of
Figure 30 beginning at steps1401. The compressed octree data is read at steps1402, and the decoder regenerates, at steps1403, the three-dimensional array from the encoded octree, as in the 2D quadtree decoding process described. Then, for any 24 bit colour value, the corresponding colour index can be determined by simply looking up the index value stored in the 3D array, as represented in steps1404. The vector post-quantisation process ends at stepsl405. This technique can be used for mapping any non-stationary three-dimensional data onto a single dimension. This is normally a requirement when vector quantisation is used to select a codebook that will be used to represent an original multi-dimensional data set. It does not matter at what stage of the process the vector quantisation is performed. For example, we could directly quadtree encode 24-bit data followed by VQ or we could VQ the data first and then quadtree encode the result as we do here. The great advantage of this method is that, in heterogeneous environments, it permits 24-bit data to be sent to clients which, if capable of displaying the 24 bit data, may do so, but, if not, may receive the pre-quantisation data and apply this to achieve real-time, high quality quantisation of the 24-bit source data.

The scene/object control data component 14 of Figure 18 permits each object to be associated with one visual data stream, one audio data stream and one of any other data streams. It also permits various rendering and presentation parameters for each object to be dynamically modified from time to time throughout the scene. These include the amount of object transparency, object scale, object volume, object position in 3D space, and object orientation (rotation) in 3D space.

The compres define object rendering transformations, animations and actions to be executed by the object control engine, interactive object behaviours, dynamic media composition parameters and conditions for execution or application of any of the preceding, for either individual objects or for entire scenes being viewed. The data packets contain the compressed information that makes up each media object. The format definition packets (DEFN) convey the configuration parameters to each codec, and specify both the format of the media objects and how the relevant data packets are to be interpreted. The scene definition packet defines the scene format, specifies the number of objects, and defines other scene properties. The USERCTRL packets are used to convey user interaction and data back to a remote server using abackchannel, the METADATA packets contain metadata about the video, the DIRECTORY packets contain information to assist random access into the bit stream, and the STREAMEND packets demarcate stream boundaries.

Access Control and Identification
Another component of the object oriented video system is means for encrypting/decrypting the video stream for security of content. The key to perform the decryption is separately and securely delivered to the end user by encoding it using the
RSA public key system.

An additional security measure is to include a universally unique brand/identifier in an encoded video stream. This takes at least four principal forms: a. In a videoconferencing application, a single unique identifier is applied to all instances of the encoded video streams b. In broadcast video-on-demand (VOD) with multiple video objects in each video data stream, each separate video object has a unique identifier for the particular video streamc. A wireless, ultrathin client system has a unique identifier which identifies the encoder type as used for wireless ultrathin system server encoding, as well as identifying a unique instance of this software encoder. d. A wireless ultrathin client system has a unique identifier that uniquely identifies the client decoder instance in order to match the Internet-based user profile to determine the associated client user.

The ability to uniquely identify a video object and data stream is particularly advantageous. In videoconference applications, there is no real need to monitor or log the teleconference video data streams, except where advertising content occurs (which is uniquely identified as per the VOD). The client side decoder software logs viewed decoded video streams (identifier, duration). Either in real-time or at subsequent synchronisation, this data is transferred to an Internet-based server. This information is used to generate marketing revenue streams as well as market research/statistics in conjunction with client personal profiles.

In VOD, the decoder can be restricted to decode broadcast streams or video only when enabled by a security key. Enabling can be performed, either in real-time if connected to the Internet, or at a previous synchronisation of the device, when accessing an Internet authentication/access/billing service provider which provides means for enabling the decoder through authorised payments. Alternatively, payments may be made for previously viewed video streams. Similarl to the advertising video streams in the video conferencing, the decoder logs VOD-related encoded video streams along with the duration of viewing. This information is transferred back to the Internet server for market research/feedback and payment purposes.

In the wireless ultrathin client (NetPC) application, real-time encoding, transmission and decoding of video streams from Internet or otherwise based computer servers is achieved by adding a unique identifier to the encoded video streams. The client-side decoder is enabled in order to decode the video stream. Enabling of the client-side decoder occurs along the lines of the authorised payments in the VOD application or through a secure encryption key process that enables various levels of access to wireless NetPC encoded video streams. The computer server encoding software facilitates multiple access levels. In the broadest form, wireless Internet connection includes mechanisms for monitoring client connections through decoder validation fed back from the client decoder software to the computer servers. These computer servers monitor client usage of server application processes and charge accordingly, and also monitor streamed advertising to end clients.

Interactive Audio Visual Markup Language (IAVML)
A powerful component of this system is the ability to control audio-visual scene composition through scripting. With scripts, the onlyconstraints on the composition functions are imposed by the limitations of the scripting language. The scripting language used in this case is IAVML which is derived from the XML standard. IAVML is the textual form for specifying the object control information that is encoded into the compressed bit stream.

IAVML is similar in some respects to HTML, but is specifically designed to be used with object oriented multimedia spatio-temporal spaces such as audio/video. It may be used to define the logical and layout structure of these spaces, including hierarchies, it may also be used to define linking, addressing and also metadata. This is achieved by permitting five basic types of markup tags to provide descriptive and referential information, etc. These are system tags, structural definition tags, presentation formatting, and links and content.

Like HTML, IAVML is not case sensitive, and each tag comes in opening and closing forms which are used to enclose the parts of the text being annotated. For example:
< TAG > some text in here < /TAG >
Structural definition of audio-visual spaces uses structural tags and include the following:
EMI66.1

< SCENE > <SEP> Defines <SEP> video <SEP> scenes
<tb> < STREAMEND > <SEP> Demarcate <SEP> streams <SEP> within <SEP> scene
<tb> < OBJECT > <SEP> Defines <SEP> object <SEP> instance
<tb> < VIDEODAT > <SEP> Defines <SEP> video <SEP> object <SEP> data
<tb> < AUDIODAT <SEP> > <SEP> Defines <SEP> audio <SEP> object <SEP> data

<tb>
EMI87.1

< TEXTDAT > <SEP> Defines <SEP> text <SEP> object <SEP> data
<tb> < GRAFDAT > <SEP> Defines <SEP> vector <SEP> object <SEP> data
<tb> < VIDEODEFN > <SEP> Defines <SEP> video <SEP> data <SEP> format
<tb> < AUDIODEFN > <SEP> Defines <SEP> audio <SEP> data <SEP> format
<tb> < METADATA > <SEP> Defines <SEP> metadata <SEP> about <SEP> given <SEP> object
<tb> < DIRECTORY > <SEP> Defines <SEP> directory <SEP> object
<tb> < OBJCONTROL > <SEP> Defines <SEP> object <SEP> control <SEP> data
<tb> < FRAME > <SEP> Defines <SEP> video <SEP> frame
<tb>
The structure defined by these tags in conjunction with the directory and meta data tags permit flexible access to and browsing of the object oriented video bitstreams.

Layout definition of audio-visual objects uses object control based layout tags (rendering parameters) to define the spatio-temporal placement of objects within any given scene and include the following:
EMI87.2

< SCALE > <SEP> Scale <SEP> of <SEP> visual <SEP> object
<tb> < VOLUME > <SEP> Volume <SEP> of <SEP> audio <SEP> data
<tb> ROTATION <SEP> Orientation <SEP> of <SEP> object <SEP> in <SEP> 3D <SEP> space
<tb> < POSITION > <SEP> Position <SEP> of <SEP> object <SEP> in <SEP> 3D <SEP> space
<tb> < TRANSPARENT > <SEP> Transparency <SEP> of <SEP> visual <SEP> objects
<tb> < DEPTH > <SEP> Change <SEP> object <SEP> Z <SEP> order
<tb> < TIME > <SEP> Start <SEP> time <SEP> of <SEP> object <SEP> in <SEP> scene
<tb> < PATH > <SEP> Animation <SEP> path <SEP> from <SEP> start <SEP> to <SEP> end <SEP> time
<tb>
Presentation definition of audio-visual objects uses presentation tags to define the presentation of objects (format definition) and include the following:
EMI87.3

< SCENESIZE > <SEP> Scene <SEP> spatial <SEP> size
<tb> < BACKCOLR > <SEP> Scene <SEP> background <SEP> colour
<tb>
EMI88.1

<tb> < FORECOLR > <SEP> Scene <SEP> foreground <SEP> colour
<tb> < VIDRATE > <SEP> Video <SEP> Frame <SEP> rate
<tb> < VIDSIZE > <SEP> Size <SEP> of <SEP> video <SEP> frame
<tb> < AUDRATE > <SEP> Audio <SEP> sample <SEP> rate
<tb> < AUDBPS > <SEP> Audio <SEP> sample <SEP> size <SEP> in <SEP> bits
<tb> < TXTFONT > <SEP> Text <SEP> Font <SEP> type <SEP> to <SEP> use
<tb> < TXTSIZE > <SEP> Text <SEP> font <SEP> size <SEP> to <SEP> use
<tb> < TXTSTYLE > <SEP> Text <SEP> style <SEP> (bold, <SEP> underline, <SEP> italic)
<tb>
Object behaviours and action tags encapsulate the object controls and includes the followingtypes:
EMI88.2

<tb> < JUMPTO > <SEP> Replaces <SEP> current <SEP> scene <SEP> or <SEP> object
<tb> < HYPERLINK > <SEP> Set <SEP> hyperlink <SEP> target
<tb> < OTHER > <SEP> Retarget <SEP> control <SEP> to <SEP> another <SEP> object
<tb> < PROTECT > <SEP> Limit <SEP> user <SEP> interaction
<tb> < LOOPCTRL > <SEP> Looping <SEP> object <SEP> control
<tb> < ENDLOOP > <SEP> Break <SEP> loop <SEP> control
<tb> < BUTTON > <SEP> Define <SEP> button <SEP> action
<tb> < CLEARWAITING > <SEP> Terminate <SEP> waiting <SEP> actions
<tb> < PAUSEPLAY > <SEP> Play <SEP> or <SEP> pause <SEP> video
<tb> < SNDMUTE > <SEP> Mute <SEP> sound <SEP> on/off
<tb> < SETFLAG > <SEP> Set <SEP> or <SEP> reset <SEP> system <SEP> flag
<tb> < SETTIMER > <SEP> Set <SEP> timer <SEP> value <SEP> and <SEP> Start <SEP> counting
<tb> < SENDFORM > <SEP> Send <SEP> system <SEP> flags <SEP> back <SEP> to <SEP> server
<tb> < CHANNEL > <SEP> Change <SEP> the <SEP> viewed <SEP> channel
<tb>
The hyperlink references within the file permits objects to be clicked on that invoke defined actions.

Simple video menus can be created using multiple media objects with the BUTTON, OTHER and JUMPTO tags defined with the OTHER parameter to indicate the current scene and the JUMPTO parameter indicating the new scene. A persistent menu can be created by defining the OTHER parameter to indicate the background video object and the JUMPTO parameter to indicate the replacement video object. A variety of conditions defined below can be used to customise these menus by disabling or enabling individual options.

Simple forms to register user selections can be created by using a scene that has a number of checkboxes created from 2 frame video objects. For each checkbox object, the JUMPTO and SETFLAG tags are defined. The JUMPTO tag is used to select which frame image is displayed for the object to indicate if the object is selected or not selected, and the indicated system flag registers the state of the selection. A media object defined with BUTTON and SENDFORM can be used to return the selections to the server for storage or processing.

In cases where there may be multiple channels being broadcast or multicast, the CHANNEL tag enables transitions between a unicast mode operation and a broadcast or multicast mode and back.

Conditions may be applied to behaviours and actions (object controls) before they are executed in the client. These are applied in IAVML by creating conditional expressions by using either < IF > or < SWITCH > tags. The client conditions include the following types:
EMI89.1

< PLAYING > <SEP> Is <SEP> video <SEP> currently <SEP> playing
<tb> < PAUSED > <SEP> Is <SEP> video <SEP> currently <SEP> paused
<tb> < STREAM > <SEP> Streaming <SEP> from <SEP> remote <SEP> server
<tb>
EMI90.1

<tb> < STORED > <SEP> Playing <SEP> from <SEP> local <SEP> storage
<tb> < BUFFERED > <SEP> Is <SEP> object <SEP> frame <SEP> # <SEP> buffered
<tb> < OVERLAP > <SEP> Need <SEP> to <SEP> be <SEP> dragged <SEP> onto <SEP> what <SEP> object
<tb> < EVENT > <SEP> What <SEP> user <SEP> event <SEP> needs <SEP> to <SEP> happen
<tb> < WAIT > <SEP> Do <SEP> we <SEP> wait <SEP> for <SEP> conditions <SEP> to <SEP> be <SEP> true
<tb> < USERFLAG > <SEP> Is <SEP> the <SEP> given <SEP> user <SEP> flag <SEP> set?
<tb> < TIMEUP > <SEP> Has <SEP> a <SEP> timer <SEP> expired?
<tb> < AND > <SEP> Used <SEP> to <SEP> generate <SEP> expressions
<tb> < OR > <SEP> Used <SEP> to <SEP> generate <SEP> expressions
<tb>
Conditions that may be applied at the remote server to control the dynamic media composition process include the following types:
EMI90.2

<tb> < FORMDATA > <SEP> User <SEP> returned <SEP> form <SEP> data
<tb> < USERCTRL > <SEP> User <SEP> interaction <SEP> event <SEP> has <SEP> occurred
<tb> < TIMEODAY > <SEP> Is <SEP> it <SEP> a <SEP> given <SEP> time
<tb> < DAYOFWEEK > <SEP> What <SEP> day <SEP> of <SEP> the <SEP> week <SEP> is <SEP> <
<tb> < DAYOFYEAR > <SEP> Is <SEP> it <SEP> a <SEP> special <SEP> day
<tb> < LOCATION > <SEP> Where <SEP> is <SEP> the <SEP> client <SEP> geographically
<tb> < USERTYPE > <SEP> Class <SEP> of <SEP> user <SEP> demographic?
<tb> < USERAGE > <SEP> What <SEP> is <SEP> age <SEP> of <SEP> user <SEP> (range)
<tb> < USERSEX > <SEP> What <SEP> is <SEP> the <SEP> sex <SEP> of <SEP> the <SEP> user <SEP> (M/F)
<tb> < LANGUAGE > <SEP> What <SEP> is <SEP> the <SEP> preferred <SEP> language
<tb> < PROFILE > <SEP> Other <SEP> subclasses <SEP> of <SEP> user <SEP> profile <SEP> data
<tb> < WAITEND > <SEP> Wait <SEP> for <SEP> end <SEP> of <SEP> current <SEP> stream
<tb> < AND > <SEP> Used <SEP> to <SEP> generate <SEP> expressions
<tb> < OR > <SEP> Used <SEP> to <SEP> generate <SEP> expressions
<tb>
An IAVML file will generally have one or more scenes and one script. Each scene is defined to have a determined spatial size, a default background colour and an optional background object in the following manner:
< SCENE ="sceneone" >
< SCENESIZE SX ="320", SY="240" >
< BACKCOLR ="#RRGGBB" >
< VIDEODAT SRC ="URL" >
< AUDIODAT SRC ="URL" >
< TEXTDAT > this is some text string < /a >
< /SCENE >
Alternatively, the background object may have been defined previously and then just declared in the scene:
< OBJECT ="backgrnd" >
< VIDEODAT SRC ="URL" >
< AUDIODAT SRC ="URL" >
< TEXTDAT > this is some text string < /a >
< SCALE ="2' >
< ROTATION ="90" >
< POSITION= XPOS="50"YPOS="100" >
< /OBJECT >
< SCENE >
< SCENESIZE SX ="320", SY="240" >
< BACKCOLR ="#RRGGBB" >
< OBJECT ="backgrnd" >
< /SCENE >
Scenes can contain any number of foreground objects:
< SCENE >
< SCENESIZE SX ="320", SY="240" >
< FORECOLR ="#RRGGBB" >
< OBJECT="foregndobject1". PATH ="somepath" >
< OBJECT ="foregndobject2", PATH ="someotherpath" >
< OBJECT ="foregnd~object3", PATH ="anypath" >
< /SCENE >
Paths are defined for each animated object in a scene:
< PATH = somepath > < TIME START="0", END="100" >
< POSITIONTIME=START, XPOS="0", YPOS="100" >
< POSITIONTIME=END, XPOS="0", YPOS="100" >
< INTERPOLATION= LINEAR >
< /PATH >
Using IAVML, content creators can textually create animation scripts for object oriented video and conditionally define dynamic media composition and rendering parameters.

After creation of an IAVML file, the remote server software processes the IAVML script to create the object control packets that are inserted into the composite video stream that is delivered to the media player. The server also uses the IAVML script internally to know how to respond to dynamic media composition requests mediated by user interaction returned from the client via user control packets.

Streaming Error Correction Protocol
In the case of wireless streaming, suitable network protocols are used to ensure that video data is reliably transmitted across the wireless link to the remote monitor. These may be
connection-oriented, such as TCP, or connectionless, such as UDP. The nature of the protocol will depend on the nature of the wireless network being used, the bandwidth, and the channel characteristics. The protocol performs the following functions: error control, flow control, packetisation, connection establishment, and link management.

There are many existing protocols for these purposes that have been designed for use with data networks. However, in the case of video, special attention may be required to handle errors, since retransmission of corrupted data is inappropriate due to the real-time constraints imposed by the nature of video on the reception and processing of transmitted data.

To handle this situation the following error control scheme is provided:
(1) Frames of video data are individually sent to the receiver, each with a check sum or cyclic redundancy check appended to enable the receiver to assess if the frame contains errors;
(2a) If there was no error, then the frame is processed normally;
(2b) If the frame is in error, then the frame is discarded and a status message is sent

to the transmitter indicating the number of the video frame that was in error;
(3) Upon receiving such an error status message, the video transmitter stops sending all predicted frames, and instead immediately sends the next available key frame to the receiver;
(4) After sending the key frame, the transmitter resumes sending normal inter frame coded video frames until another error status message is received.

A key frame is a video frame that has only been intra-frame coded but not inter-frame coded. Inter-frame coding is where the prediction processes are performed and makes these frames dependent on all the preceding video frames after and including the last key frame. Key frames are sent as the first frame and whenever an error occurs. The first frame needs to be a key frame because there is no previous frame to use for inter-frame coding.

Voice Command Process
Since wireless devices are small, the ability to enter text commands manually for operating the device and data processing is difficult. Voice commands have been suggested as a possible avenue for achieving hands-free operation of the device. This presents a problem in that many wireless devices have very low processing power, well below that required for general automatic speech recognition (ASR). The solution in this case is to capture the user speech on the device, compress it, and send it to the server for ASR and execution as shown in Figure31, since in any case the server will be actioning all user commands. This frees the device from having to perform this complex processing, since it is likely to be devoting most of its processing resources to decoding and rendering any streaming audio/video content. This process is depicted by the flowchart of Figure31, beginning at steps1501. The process is initiated when the user speaks a command into the device microphone at steps1502. If, at steps1503, voice commands are disabled, the voice command is ignored and the process ends at stepsl517. Otherwise, the voice command speech is captured and compressed at stepsl504, the encoded samples are inserted into USERCTRL packets at stepsl505, and sent to a voice command server at steps1506. The voice command server then performs automatic speech recognition at steps1507, and maps the transcribed speech to a command set at step <SEP> Type <SEP> Definition
<tb> BYTE <SEP> 8 <SEP> bits, <SEP> unsigned <SEP> char
<tb> WORD <SEP> 16 <SEP> bits, <SEP> unsigned <SEP> short
<tb> DWORD <SEP> 32 <SEP> bits, <SEP> unsigned <SEP> long
<tb> BYTE <SEP> [] <SEP> String, <SEP> byte <SEP> [0] <SEP> specifies <SEP> length <SEP> up <SEP> to <SEP> 254, <SEP> (255
<tb> <SEP> reserved)
<tb> IPOINT <SEP> 12bits <SEP> unsigned, <SEP> 12 <SEP> bits <SEP> unsigned, <SEP> (x, <SEP> y)
<tb> DPOINT <SEP> 8 <SEP> bits <SEP> unsigned <SEP> char, <SEP> 8 <SEP> bits <SEP> unsigned <SEP> char,
<tb> <SEP> (dx, <SEP> dy)
<tb>
The file stream is divided into packets or blocks of data. Each packet is encapsulated within a container similar to the concept of atoms in Quicktime, but is not hierarchical. A container consists of a BaseHeader record that specifies the payload type and some auxiliary packet control information and the size of the data payload. The payload type defines the various kinds of packet in the stream. The one exception to this rule is the
SystemControl packet used to perform end-to-end network link management. These packets consist of a BaseHeader with no payload. In this case, the payload size field is reinterpreted. In the case of streaming over circuit switched networks, a preliminary, additional network container is used to achieve error resilience by providing for synchronisation and checksums
There are four main types of packets within the bit stream: data packets, definition packets, control packets and metadata packets of various kinds. Definition packets are used to convey media format and codec information that is used to interpret the data packets. Data packets convey the compressed data to be decoded by the selected application. Hence an appropriate Definition packet precedes any data packets of each given data type. Control packets that define rendering and animation parameters occur after Definition but before Data Packets.

Conceptually, the object oriented data can be considered to consist of 3 main interleaved streams of data. The definition, data, control streams. The metadata is an optional fourth stream. These 3 main streams interact to generate the final audio-visual experience that is presented to a viewer.

All files start with a SceneDefinition block which defines the AV scene space into which any audio or video streams or objects will be rendered. Metadata and directory packets contain additional information about the data contained by the data and definition packets to assist browsing of the data packets. If any metadata blocks exist, they occur immediately after a SceneDefinition packet. A directory packet immediately follows a
Metadata packet or a SceneDefinition packet if there is no Metadata packet.

The file format permits integration of diverse media types to support object oriented interaction, both when streaming the data from a remote server or accessing locally stored content. To this end, multiple scenes can be defined and each may contain up to 200 separate media objects simultaneously. These objects may be of a single media type such as video, audio, text or vector graphics, or composites created from combinations of these media types.

As shown in Figure 4, the file structure defines a hierarchy of entities: a file can contain one of more scenes, each scene may contain one of more objects, and each object can contain one or more frames. In essence, each scene consists of a number of separate interleaved data streams, one for each object each consisting of a number of frames. Each stream is consists of one of more definition packets, followed by data and control packets all bearing the sameobject-id number.

Stream Syntax
Valid Packet Types
The BaseHeader allows for a total of up to 255 different packet types according to payload. This section defines the packet formats for the valid packet types as listed in the following table.
EMI119.1

<tb>

Value <SEP> Data <SEP> Type <SEP> Payload <SEP> Comment
<tb> <SEP> SCENEDEFN <SEP> SceneDefinition <SEP> Defines <SEP> scene <SEP> space <SEP> properties
<tb> <SEP> 1 <SEP> VIDEODEFN <SEP> VideoDefinition <SEP> Defines <SEP> video <SEP> format/codec <SEP> properties
<tb> <SEP> 2 <SEP> AUDIODEFN <SEP> AudioDefinition <SEP> Defines <SEP> audio <SEP> format/codec <SEP> properties
<tb> <SEP> 3 <SEP> TEXTDEFN <SEP> TextDefinition <SEP> Defines <SEP> text <SEP> format/codec <SEP> properties
<tb> <SEP> 4 <SEP> GRAFDEFN <SEP> GrafDefinition <SEP> Defines <SEP> vector <SEP> graphics <SEP> format/codec
<tb> <SEP> properties
<tb> <SEP> 5 <SEP> VIDEOKEY <SEP> VideoKey <SEP> Video <SEP> Key <SEP> Frame <SEP> data
<tb> <SEP> 6 <SEP> VIDEODAT <SEP> VideoData <SEP> Compressed <SEP> Video <SEP> data
<tb> <SEP> 7 <SEP> AUDIODAT <SEP> AudioData <SEP> Compressed <SEP> audio <SEP> data
<tb> <SEP> 8 <SEP> TEXTDAT <SEP> TextData <SEP> Text <SEP> data
<tb> <SEP> 9 <SEP> GRAFDAT <SEP> GrafData <SEP> Vector <SEP> Graphics <SEP> data
<tb> <SEP> 10 <SEP> MUSICDAT <SEP> Music <SEP> Data <SEP> Music <SEP> Score <SEP> Data
<tb> <SEP> 11 <SEP> OBJCTRL <SEP> ObjectControl <SEP> Defines <SEP> object <SEP> animation/rendering
<tb> <SEP> properties

<tb> <SEP> 12 <SEP> LINKCTRL <SEP> Used <SEP> for <SEP> streaming <SEP> end <SEP> to <SEP> end <SEP> link
<tb> <SEP> management
<tb> <SEP> 13 <SEP> USERCTRL <SEP> UserControl <SEP> Back <SEP> channel <SEP> for <SEP> user <SEP> system <SEP> interaction
<tb> <SEP> 14 <SEP> METADATA <SEP> MetaData <SEP> Contains <SEP> meta <SEP> data <SEP> about <SEP> AV <SEP> scene
<tb> <SEP> 15 <SEP> DIRECTORY <SEP> Directory <SEP> Directory <SEP> of <SEP> data <SEP> or <SEP> system <SEP> objects,
<tb> <SEP> 16 <SEP> VIDEOENH-RESERVED-video <SEP> enhancement <SEP> data
<tb> <SEP> 17 <SEP> AUDIOENH-RESERVED-audio <SEP> enhancement <SEP> data
<tb> <SEP> 18 <SEP> VIDEOEXTN-Redundant <SEP> I <SEP> frames <SEP> for <SEP> error <SEP> correction
<tb> <SEP> 19 <SEP> VIDEOTERP <SEP> Video <SEP> Data <SEP> Discardable <SEP> Interpolated <SEP> video <SEP> files
<tb>
EMI120.1

<tb> <SEP> 20 <SEP> STREAMEND-Indicates <SEP> end <SEP> of <SEP> stream <SEP> and <SEP> the <SEP> start <SEP> of <SEP> a
<tb> <SEP> new <SEP> stream
<tb> 21 <SEP> MUSICDEFN <SEP> Music <SEP> Defin <SEP> Defines <SEP> music <SEP> format
<tb> 22 <SEP> FONTLIB <SEP> FontLibDefn <SEP> font <SEP> library <SEP> data
<tb> 23 <SEP> OBJLIBCTRL <SEP> ObjectLibCntrol <SEP> object/font <SEP> library <SEP> control
<tb> 255--RESERVED
<tb>
BaseHeader
Short BaseHeader is for packets that are shorter than 65536 bytes
EMI120.2

Description <SEP> Type <SEP> Comment
<tb> Type <SEP> BYTE <SEP> Payload <SEP> packet <SEP> type <SEP> [0], <SEP> can <SEP> be <SEP> definition, <SEP> data <SEP> or <SEP> control
<tb> <SEP> packet
<tb> Obj~id <SEP> BYTE <SEP> Object <SEP> stream <SEP> ID-what <SEP> object <SEP> does <SEP> this <SEP> belong <SEP> to
<tb> Seq~no <SEP> WORD <SEP> Frame <SEP> sequence <SEP> number, <SEP> individual <SEP> sequence <SEP> for <SEP> each <SEP> object
<tb> Length <SEP> WORD <SEP> Size <SEP> of <SEP> frame <SEP> to <SEP> follow <SEP> in <SEP> bytes <SEP> {0 <SEP> means <SEP> end <SEP> of <SEP> stream}
<tb>
Long BaseHeader will support packets from 64K up toOxFFFFFFFF bytes
EMI120.3

<tb> Description <SEP> Type <SEP> Comment
<tb> <SEP> Type <SEP> BYTE <SEP> Payload <SEP> packet <SEP> type <SEP> [0], <SEP> can <SEP> be <SEP> definition, <SEP> data <SEP> or <SEP> control
<tb> <SEP> packet
<tb> <SEP> Objid <SEP> BYTE <SEP> Object <SEP> stream <SEP> ID-what <SEP> object <SEP> does <SEP> this <SEP> belong <SEP> to
<tb> <SEP> Seq~no <SEP> WORD <SEP> Frame <SEP> sequence <SEP> number, <SEP> individual <SEP> sequence <SEP> for <SEP> each <SEP> object
<tb> <SEP> Flag <SEP> WORD <SEP> OxFFFF
<tb> <SEP> Length <SEP> DWOR <SEP> Size <SEP> of <SEP> frame <SEP> to <SEP> follow <SEP> in <SEP> bytes
<tb> <SEP> D
<tb>
System BaseHeader is for end-to-end network link management
EMI120.4

<tb> Description <SEP> Type <SEP> Comment
<tb>
EMI121.1

Type <SEP> BYTE <SEP> DataType <SEP> = <SEP> SYSCTRL
<tb> Objid <SEP> BYTE <SEP> Object <SEP> stream <SEP> ID-what <SEP> object <SEP> does <SEP> this <SEP> belong <SEP> to
<tb> Seq~no <SEP> WORD <SEP> Frame <SEP> sequence <SEP> number, <SEP> individual <SEP> sequence <SEP> for <SEP> each <SEP> object
<tb> Status <SEP> WORD <SEP> StatusType <SEP> {ACK, <SEP> NAK, <SEP> CONNECT, <SEP> DISCONNECT,
<tb> <SEP> IDLE} <SEP> +object <SEP> type
<tb>
Total size is 6 or 10 bytes
EMI121.2

<tb> SceneDefinition
<tb> <SEP> Description <SEP> Type <SEP> Comment
<tb> <SEP> Magic <SEP> BYTE <SEP> [4] <SEP> ASKY <SEP> = <SEP> Ox41534B59 <SEP> (used <SEP> for <SEP> format <SEP> validation)
<tb> <SEP> Version <SEP> BYTE <SEP> Version <SEP> OxOO-current
<tb> <SEP> Compatible <SEP> BYTE <SEP> Version <SEP> 0x00-current <SEP> - <SEP> minimum <SEP> format <SEP> playable
<tb> <SEP> Width <SEP> WORD <SEP> SceneSpace <SEP> width <SEP> (0=unspecified)
<tb> <SEP> Height <SEP> WORD <SEP> SceneSpace <SEP> height <SEP> (0 <SEP> = <SEP> unspecified)
<tb> <SEP> BackFill <SEP> WORD <SEP> RESERVED-Scene <SEP> Fill <SEP> Style/colour
<tb> <SEP> NumObjs <SEP> BYTE <SEP> How <SEP> many <SEP> objects <SEP> in <SEP> this <SEP> scene
<tb> <SEP> Mode <SEP> BYTE <SEP> Frame <SEP> playout <SEP> mode <SEP> bitfield
<tb>
Total size is 14 bytes
MetaData
EMI121.3

<tb> DescriptionType <SEP> Comment
<tb> <SEP> NumItem <SEP> WORD <SEP> Number <SEP> of <SEP> scenes/frames <SEP> in <SEP> file/scene <SEP> (0 <SEP> = <SEP> unspecified)
<tb> <SEP> SceneSize <SEP> DWORD <SEP> Size <SEP> in <SEP> bytes <SEP> of <SEP> file/scene/object <SEP> including <SEP> (0 <SEP> =
<tb> <SEP> unspecified)
<tb> <SEP> SceneTime <SEP> WORD <SEP> Playing <SEP> time <SEP> of <SEP> file/scene/object <SEP> in <SEP> seconds <SEP> (0 <SEP> =
<tb> <SEP> unspecified/static)
<tb> <SEP> BitRate <SEP> WORD <SEP> Bit <SEP> rate <SEP> of <SEP> this <SEP> file/scene/object <SEP> in <SEP> kbits/sec
<tb> <SEP> MetaMask <SEP> DWORD <SEP> Bit <SEP> field <SEP> specifying <SEP> what <SEP> optional <SEP> 32 <SEP> meta <SEP> data <SEP> tags
<tb> <SEP> follow.
<tb>

<SEP> Title <SEP> BYTE <SEP> [] <SEP> Title <SEP> of <SEP> video <SEP> file/scene-whatever <SEP> you <SEP> like, <SEP> byte <SEP> [O] <SEP> =
<tb>
EMI122.1

<tb> <SEP> length
<tb> Creator <SEP> BYTE <SEP> [] <SEP> Who <SEP> created <SEP> this, <SEP> byte <SEP> [0] <SEP> = <SEP> length
<tb> Date <SEP> BYTE <SEP> [8] <SEP> Creation <SEP> date <SEP> in <SEP> ASCII <SEP> = > <SEP> DDMMYYYY
<tb> Copyright <SEP> BYTE <SEP> []
<tb> Rating <SEP> BYTE <SEP> X, <SEP> XX, <SEP> XXX <SEP> etc
<tb> EncoderID <SEP> BYTE <SEP> []
<tb> -BYTE
Directory
This is an array of type WORD or DWORD. The size is given by the Length field in the BaseHeader packet.

VideoDefinition
EMI122.2

<SEP> Description <SEP> Type <SEP> Comment
<tb> Codec <SEP> BYTE <SEP> Video <SEP> Codec <SEP> Type <SEP> {RAW, <SEP> QTREE} <SEP> ;
<tb> Frate <SEP> BYTE <SEP> Frame <SEP> rate <SEP> {O <SEP> = <SEP> stop/pause <SEP> video <SEP> play} <SEP> in <SEP> 1/5 <SEP> sec
<tb> Width <SEP> WORD <SEP> Width <SEP> Of <SEP> video <SEP> frame
<tb> Height <SEP> WORD <SEP> Height <SEP> Of <SEP> video <SEP> frame
<tb> Time <SEP> DWORD <SEP> Time <SEP> stamp <SEP> in <SEP> 50ms <SEP> resolution <SEP> from <SEP> start <SEP> of <SEP> scene <SEP> (0 <SEP> =
<tb> <SEP> unspecified)
<tb>
Total size is 10 bytes
EMI123.1

<tb> AudioDefinition
<tb> <SEP> Description <SEP> Type <SEP> Comment
<tb> <SEP> Codes <SEP> BYTE <SEP> Audio <SEP> Codec <SEP> Codec <SEP> Type <SEP> {RAW, <SEP> G723,,ADPCM}
<tb> <SEP> Format <SEP> BYTE <SEP> Audio <SEP> Format <SEP> in <SEP> bits <SEP> 7-4, <SEP> Sample <SEP> Rate <SEP> in <SEP> bits <SEP> 3-0
<tb> <SEP> Fsize <SEP> WORD <SEP> Samples <SEP> per <SEP> frame
<tb> <SEP> Time <SEP> DWORD <SEP> Time <SEP> stamp <SEP> in <SEP> 50ms <SEP> resolution <SEP> from <SEP> start <SEP> of <SEP> scene <SEP> (0 <SEP> =
<tb> <SEP> unspecified)
<tb>
Total size is 8 bytesTextDefinition
EMI123.2

<tb> Description <SEP> Type <SEP> Comment
<tb> Type <SEP> BYTE <SEP> Type <SEP> in <SEP> low <SEP> nibble <SEP> {TEXT, <SEP> HTML, <SEP> etc} <SEP> compression <SEP> in
<tb> <SEP> high <SEP> nibble
<tb> Fontinfo <SEP> BYTE <SEP> Font <SEP> size <SEP> in <SEP> low <SEP> nibble, <SEP> Front <SEP> Style <SEP> in <SEP> high <SEP> nibble
<tb> Colour <SEP> WORD <SEP> Font <SEP> colour
<tb> BackFill <SEP> WORD <SEP> Background <SEP>
GrafDefinition
EMI124.1

<tb> Description <SEP> Type <SEP> Comment
<tb> Xpos <SEP> WORD <SEP> XPos <SEP> relative <SEP> to <SEP> object <SEP> origin <SEP> if <SEP> defined <SEP> relative <SEP> to <SEP> 0,0
<tb> <SEP> otherwise
<tb> Ypos <SEP> WORD <SEP> XPos <SEP> relative <SEP> to <SEP> object <SEP> origin <SEP> if <SEP> defined <SEP> relative <SEP> to <SEP> 0, <SEP> 0
<tb> <SEP> otherwise
<tb> FrameRate <SEP> WORD <SEP> Frame <SEP> delay <SEP> in <SEP> 8.8 <SEP> fps
<tb> FrameSize <SEP> WORD <SEP> RESERVED <SEP> Frame <SEP> size <SEP> in <SEP> twips <SEP> (1/20 <SEP> pel)-used <SEP> for
<tb> <SEP> scaling <SEP> to <SEP> fit <SEP> scene <SEP> space
<tb> Time <SEP> DWORD <SEP> Time <SEP> stamp <SEP> in <SEP> 50ms <SEP> resolution <SEP> from <SEP> start <SEP> of <SEP> scene
<tb>

EMI124.2

<tb> Total <SEP> size <SEP> is <SEP> 12 <SEP> bytes
<tb>
VideoKey, VideoData, AudioData, TextData, GrafData and MusicData
EMI124.3

Description <SEP> Type <SEP> Comment
<tb> Payload <SEP> - <SEP> Compressed <SEP> data
<tb>
StreamEnd
EMI124.4

<SEP> r
<tb> Description <SEP> Type <SEP> Comment
<tb> <SEP> StreamObjs <SEP> BYTE <SEP> How <SEP> many <SEP> objects <SEP> interleaved <SEP> in <SEP> the <SEP> next <SEP> stream
<tb> <SEP> StreamMode <SEP> BYTE <SEP> RESERVED
<tb> <SEP> ize <SEP> DWORD <SEP> Length <SEP> of <SEP> next <SEP> stream <SEP> in <SEP> bytes
<tb>
Total size is 6 bytesUscrControl
EMI124.5

<SEP> Description <SEP> Type <SEP> Comment
<tb> Event <SEP> BYTE <SEP> User <SEP> data <SEP> Type <SEP> eg. <SEP> PENDOWN, <SEP> KEYEVENT,
<tb> <SEP> PLAYCTRL,
<tb>
EMI125.1

Key <SEP> BYTE <SEP> Parameter <SEP> 1 <SEP> = <SEP> Keycode <SEP> value/Start/Stop/Pause
<tb> HiWord <SEP> WORD <SEP> Parameter <SEP> 2 <SEP> =X <SEP> position
<tb> LoWord <SEP> WORD <SEP> Parameter <SEP> 3 <SEP> =Y <SEP> position
<tb> Time <SEP> WORD <SEP> Timestamp <SEP> = <SEP> sequence <SEP> number <SEP> of <SEP> activated <SEP> object
<tb> Data <SEP> BYTE <SEP> [] <SEP> * <SEP> Optional <SEP> field <SEP> for <SEP> form <SEP> field <SEP> data
<tb>
Total size is 8+ bytesOb jectControl
EMI125.2

<tb> Description <SEP> Type <SEP> Comment
<tb> ControlMask <SEP> BYTE <SEP> Bit <SEP> field <SEP> defining <SEP> common <SEP> object <SEP> controls
<tb> ControlObject <SEP> BYTE <SEP> (optional) <SEP> ID <SEP> of <SEP> affected <SEP> object
<tb> Timer <SEP> WORD <SEP> (optional) <SEP> Top <SEP> nibble=timer <SEP> number, <SEP> bottom <SEP> 12
<tb> <SEP> bits <SEP> = <SEP> 100ms <SEP> steps
<tb> ActionMask <SEP> WORD <SEP> I <SEP> BYTE <SEP> Bit <SEP> field <SEP> actions <SEP> defined <SEP> in <SEP> remaining <SEP> payload
<tb> Params <SEP> Parameters <SEP> for <SEP> actions <SEP> defined <SEP> by <SEP> Action <SEP> bit <SEP> field
<tb>
ObjLibCtrl
EMI125.3

<tb> Description <SEP> Type <SEP> Comment
<tb> Action <SEP> BYTE <SEP> What <SEP> to <SEP> do <SEP> with <SEP> this <SEP> object
<tb> <SEP> 1. <SEP> INSERT-does <SEP> not <SEP> overwrite <SEP> LibID <SEP> location
<tb> <SEP> 2. <SEP> UPDATE-overwrites <SEP> into <SEP> LibID <SEP> location
<tb> <SEP> 3. <SEP> PURGE-removes
<tb> <SEP> 4. <SEP> QUERY-returns <SEP> LibID/Version <SEP> for <SEP> Unique <SEP> nid
<tb> <SEP> object
<tb> LibID <SEP> BYTE <SEP> Object's <SEP> index/number <SEP> in <SEP> the <SEP> library
<tb> Version <SEP> BYTE <SEP> this <SEP> object's <SEP> version <SEP> number
<tb> Persist/Expi <SEP> BYTE <SEP> Does <SEP> this <SEP> get <SEP> garbage <SEP> collected <SEP> or <SEP> does <SEP> it <SEP> stick <SEP> around, <SEP> 0
<tb> re <SEP> = <SEP> remove <SEP> after <SEP> session, <SEP> 1-254 <SEP> = <SEP> days <SEP> before <SEP> expiry, <SEP> 255
<tb> <SEP> = <SEP> persist
<tb> Access <SEP> BYTE <SEP> Access <SEP> control <SEP> function
<tb> <SEP> Top <SEP> 4 <SEP> bits: <SEP> Who <SEP> can <SEP> overwrite <SEP> or <SEP> remove <SEP> this <SEP> object,
<tb> <SEP> 1. <SEP> any <SEP> session <SEP> at <SEP> will <SEP> (by <SEP> LibID)
<tb> <SEP> 2. <SEP> system <SEP> purge/reset
<tb> <SEP> 3. <SEP> by <SEP> knowning <SEP> the <SEP> unique <SEP> ID/libID <SEP> for <SEP> object
<tb>
EMI126.1

<tb> <SEP> 4. <SEP> never/RESERVED
<tb> <SEP> Bit <SEP> 3: <SEP> Can <SEP> the <SEP> user <SEP> transfer <SEP> this <SEP> object <SEP> to <SEP> another,
<tb> <SEP> beaming(1=YES)
<tb> <SEP> Bit <SEP> 2: <SEP> Can <SEP> the <SEP> user <SEP> directly <SEP> play <SEP> this <SEP> from <SEP> the <SEP> library
<tb> <SEP> (Yes=t/No)
<tb> <SEP> Bit <SEP> 1: <SEP> RESERVED
<tb> <SEP> Bit <SEP> 0: <SEP> RESERVED
<tb> UniqueID <SEP> BYTE <SEP> [] <SEP> Unqiue <SEP> ID/label <SEP> for <SEP> this <SEP> object
<tb> State <SEP> DWORD?? <SEP> Where <SEP> did <SEP> you <SEP> get <SEP> it <SEP> from/how, <SEP> many <SEP> hops, <SEP> feeding <SEP> time
<tb> <SEP> ?? <SEP> else <SEP> it <SEP> dies

<tb> <SEP> 1. <SEP> Hop <SEP> count
<tb> <SEP> 2. <SEP> Source <SEP> (SkyMail, <SEP> SkyFile, <SEP> SkyServer)
<tb> <SEP> 3. <SEP> time <SEP> since <SEP> activation
<tb> <SEP> 4. <SEP> # <SEP> Activations
<tb> Semantics
BaseHeader
This is the container for all information packets in the stream.

Type-BYTE
Description-Specifies the type of payload in packet as defined above
Valid Values: enumerated0-255, see Payload type table belowObjid-BYTE
Description-Object ID-defines scope-what object does this packet belong to.

Also defines the Z-order in steps of 255, that increases towards the viewer.

Up to four different media types can share the sameobjid.

Valid Values: 0-NumObjs (max 200) NumObjs defined in SceneDefinition
201-253: Reserved for system use
250: Object Library
25 1 : RESERVED
252: Directory of Streams
253: Directory of Scenes
254: This Scene
255: This File
Seqno-WORD
Description-Frame sequence number, individual sequence for each media type within an object. Sequence number are restarted after each new SceneDefinition packet.

Valid Values: 0-OxFFFF
Flag (optional)-WORD
Description-Used to indicate long baseheader packet.

Valid Values:OxFFFF
Length-WORD/DWORD
Used to indicate payload length in bytes, (if flag set packet size = length +OxFFFF).

Valid Values: Ox0001-OxFFF, If flag is set0x00000001-OxFFFFFFFF
0-RESERVED forEndof File/Stream OxFFFF Status-WORD
Used with SysControl DataType flag, for end to end link management.

Valid Values: enumerated0-65535
EMI128.1

<tb> Value <SEP> Type <SEP> Comment
<tb> 0 <SEP> ACK <SEP> Acknowledge <SEP> packet <SEP> with <SEP> given <SEP> obj~id <SEP> and
<tb> <SEP> seqno
<tb> 1 <SEP> NAK <SEP> Flag <SEP> error <SEP> on <SEP> packet <SEP> with <SEP> given <SEP> obj~id <SEP> and
<tb> <SEP> seq~no
<tb> 2 <SEP> CONNECT <SEP> Establish <SEP> client/server <SEP> connection
<tb> 3 <SEP> DISCONNECT <SEP> Break <SEP> client/server <SEP> connection
<tb> 4 <SEP> IDLE <SEP> Link <SEP> is <SEP> idle
<tb> 5-65535-RESERVED
<tb>
SceneDefinition
This defines the properties of the AV scene space into which the video and audio objects will be played.

Magic-BYTE [4]
Description-used for format validation,
Valid Value: ASKY =Ox41534B59
Version-BYTE
Description-used for stream format validation
Valid Range: 0-255 (current =0)
Compatible-BYTE
Description-what is the minimum player that can read this format
Valid Range: 0-Version
Width-WORD
Description-SceneSpace width in pixels
Valid Range:OxOOOO-OxFFFF
Height-WORD
Description-SceneSpace height in pixels
Valid Range:OxOOOO-OxFFFF
BackFill- (RESERVED) WORD
Description-background scene fill (bitmap, solid colour, gradient)
Valid Range:0x1000-OxFFFF solid colour in 15 bit format
else the low order BYTE defines the object id for a vector object
and the high order BYTE(0-15) is an index to gradient fill style table
This vector object definition occurs prior to any data control packets
NumObjs-BYTE
Description-how many data objects are in this scene
Valid Range: 0-200 (201-255 reserved for system objects)
Mode-BYTE
Description-Frame playout mode bitfield
Bit: [7] play status-paused = 1, play= 0//continuous play or step through
Bit: [6] RESERVED Zooming-prefer = 1, normal = 0//play zoomed
Bit: [5] RESERVED-datastorage-live 1, stored = 0 H being streamed?
Bit: [4] RESERVEDstreaming-reliable = 1, best try =0//is streaming reliable
Bit: [3] RESERVED data source-video = 1, thinclient = 0//originating source
Bit: [2] RESERVED Interaction-allow = 1, disallow =0
Bit: [1] RESERVED
Bit:[0] Library Scene-is this a library scenel=yes, 0= no

MetaData
This specifies metadata associated with either an entire file, scene or an individual AV object. Since files can be concatenated, there is no guarantee that a metadata block with file scope is valid past the last scene it specifies. Simply comparing the file size with the SCENESIZE field in this Metadata packet however can validate this.

The OBJID field in baseHeader defines the scope of a metadata packet. This scope can be the entire file (255), a single scene (254), or an individual video object (0-200). Hence if MetaData packets are present in a file they occur in flocks (packs?) immediately following SceneDefinition packets.

NumItem-WORD
Description-Number of scenes/frames in file/scene,
For scene scopeNumItem contains the number of frames for video object withobjid=0
Valid Range:0-65535 (0 = unspecified)
SceneSize-DWORD
Description-Self inclusive size in bytes of file/scene/object including,
Valid Range: OxOOOO-OxFFFFFFFF(0 = unspecified)
SceneTime-WORD
Description-Playing time offile/scene/object in seconds,
Valid Range: OxOOOO-OxFFFF(0 = unspecified)
BitRate-WORD
Description-bit rate of this file/scene/object in kbits/sec,
Valid Range: OxOOOO-OxFFFF(0 = unspecified)
MetaMask- (RESERVED) DWORD
Description-Bit field specifying what optional 32 meta data fields follow in order,
Bit Value[31] : Title
Bit Value [30]: Creator
Bit Value [29]: Creation Date
Bit Value [28]: Copyright
Bit Value [27]: Rating
Bit Value [26]:EncoderID
Bit Value [26-27]: RESERVED
Title- (Optional) BYTE []
Description-String of up to 254 chars
Creator- (Optional) BYTE []
Description-String of up to 254 chars
Date- (Optional) BYTE [8]
Description-Creation date in ASCII= > DDMMYYYY
Copyright- (Optional) BYTE []
Description-String of up to 254 chars
Rating- (Optional) BYTE
Description-BYTE specifying 0-255
Directory
This specifies directory information for an entire file or for a scene. Since the files can be concatenated, there is no guarantee that a metadata block with file scope is valid past the last scene it specifies. Simply comparing the file size with the SCENESIZE field in a Metadata packet however can validate this.

The OBJID field in baseHeader defines the scope of a directory packet. If the value of the
OBJID field is less than 200 then the directory is a listing of sequence numbers (WORD) for keyframes in a video data object. Else, the directory is a location table of system objects. In this case the table entries are relative offset in bytes (DWORD) from the start of the file (for directories of scenes and directories) or scene for other system objects). The number of entries in the table and the table size can be calculated from the LENGTH field in the BaseHeader packet.

Similar to MetaData packets if Directory packets are present in a file they occur in flocks (packs?) immediately following SceneDefinition, or Metadata packets.

VideoDefinition
Codec-BYTE
Description-Compression Type
Valid Values: enumerated 0-255
EMI132.1

<tb> Value <SEP> Codec <SEP> Comment
<tb> 0 <SEP> RAW <SEP> Uncompressed, <SEP> the <SEP> first <SEP> byte <SEP> defines <SEP> colour <SEP> depth
<tb> 1 <SEP> QTREE <SEP> Default <SEP> Video <SEP> codec
<tb> 2-255-RESERVED
<tb>
Frate-BYTE
Description-frame playout rate in1/5 sec (ie max = 51 fps, min = 0.2 fps)
Valid Values:1-255, play/start playing if stopped
0-stop playing
Width-WORD
Description-how wide in pixels in video frame
Valid Values: 0-65535
Height-WORD
Description-how high in pixels in video frame
Valid Values: 0-65535
Times-WORD
Description-Time stamp in50ms resolution from start of scene(0 = unspecified)
Valid Values:1-OxFFFFFFFF (0 = unspecified) AudioDefinition
Codec-BYTE
Description-Compression Type
Valid Values: enumerated 1(0 = unspecified)
EMI133.1

<tb> <SEP> Comment
<tb> 0 <SEP> WAV <SEP> Uncompressed
<tb> 1 <SEP> G723 <SEP> Default <SEP> Video <SEP> codec
<tb> 2 <SEP> IMA <SEP> Interactive <SEP> Multimedia <SEP> Association <SEP> ADPCM
<tb> 3-255-RESERVED
<tb>

Format-BYTE
Description-This BYTE is split into 2 separate fields that are independently defined. The top 4 bits define the audio format (Format > > 4) while the bottom 4 bits separate define the sample rate (Format & OxOF).

Low 4 Bits, Value: enumerated0-15, Sampling Rate
EMI134.1

<tb> Value <SEP> Samp.Rate <SEP> Comment
<tb> 0 <SEP> 0 <SEP> 0-stop <SEP>
<tb> 1 <SEP> 5. <SEP> 5 <SEP> kHz <SEP> 5. <SEP> 5kHz <SEP> Very <SEP> low <SEP> rate <SEP> sampling, <SEP> start <SEP> playing <SEP> if
<tb> <SEP> stopped
<tb> 2 <SEP> 8 <SEP> kHz <SEP> Standard <SEP> 8000 <SEP> Hz <SEP> Sampling, <SEP> start <SEP> playing <SEP> if <SEP> stopped
<tb> 3 <SEP> 11 <SEP> kHz <SEP> Standard <SEP> 11025 <SEP> Hz <SEP> Sampling, <SEP> start <SEP> playing <SEP> if <SEP> stopped
<tb> 4 <SEP> 16 <SEP> kHz <SEP> 2x <SEP> 8000 <SEP> Hz <SEP> Sampling, <SEP> start <SEP> playing <SEP> if <SEP> stopped
<tb> 5 <SEP> 22 <SEP> kHz <SEP> Standard <SEP> 22050 <SEP> Hz <SEP> Sampling, <SEP> start <SEP> playing <SEP> if <SEP> stopped
<tb> 6 <SEP> 32 <SEP> kHz <SEP> 4x <SEP> 8000 <SEP> Hz <SEP> Sampling, <SEP> start <SEP> playing <SEP> if <SEP> stopped
<tb> 7 <SEP> 44 <SEP> kHz <SEP> Standard <SEP> 44100 <SEP> Hz <SEP> Sampling, <SEP> start <SEP> playing <SEP> if <SEP> stopped
<tb> 8-15 <SEP> RESERVED
<tb>
Bits 4-5, Value: enumerated 0-3, Format
EMI134.2

<tb> Value <SEP> Format <SEP> Comment
<tb> 0 <SEP> MON08 <SEP> Monophonic, <SEP> 8 <SEP> bits <SEP> per <SEP> sample
<tb> 1 <SEP> MON016 <SEP> Monophonic, <SEP> 16 <SEP> bits <SEP> per <SEP> sample
<tb> 2 <SEP> STERE08 <SEP> Stereophonic, <SEP> 8 <SEP> bits <SEP> per <SEP> sample
<tb> 3 <SEP> STERE016 <SEP> Stereophonic, <SEP> 16 <SEP> bits <SEP> per <SEP> sample
<tb>
High 2 Bits (6-7), Value: enumerated 0-3, Special
EMI134.3

<tb> <SEP> Codec <SEP> Comment
<tb> WAV <SEP> RESERVED <SEP> (unused)
<tb> G. <SEP> 723 <SEP> RESERVED <SEP> (unused)
<tb> IMA <SEP> Bits <SEP> Per <SEP> Sample <SEP> (Value <SEP> + <SEP> 2)
<tb>
Fsize-WORD
Description-samples per frame
Valid Values: 0-65535
Times-WORD
Description-Time stamp in50ms resolution from start of scene(0 = unspecified)
Valid Values:1-OxFFFFFFFF (0 = unspecified)
TextDefinition
We need to include writing direction, it can be LRTB, or RLTB or TBRL or TBLR. This can be done by using a special letter code in the body of the text to indicate the direction, for example we could use DC1-DC4 (ASCII device control codes 17-20) for this task
We also need to have a font table downloaded at the start with bitmap fonts. Depending on the platform the player is running on the renderer may either ignore the bitmap fonts or attempt to use the bitmap fonts for rendering the text. If there is no bit map font table or if it being ignored by the player then the rendering system will automatically attempt to use the Operating System text output functions to render the text.

Type-BYTE
Description-Defines how text data is interpreted in low nibble (Type & OxOF) and compression method in high nibble (Type 4)
Low 4 Bits, Value: enumerated0-15, Type-interpretation
EMI135.1

<tb> Value <SEP> Type <SEP> Comment
<tb> <SEP> 0 <SEP> PLAIN <SEP> Plain <SEP> text-no <SEP> interpretation
<tb> <SEP> 1 <SEP> TABLE <SEP> RESERVED-table <SEP> data
<tb> <SEP> 2 <SEP> FORM <SEP> Form/Text <SEP> Field <SEP> for <SEP> user <SEP> input
<tb> <SEP> 3 <SEP> WML <SEP> RESERVED <SEP> WAP-WML
<tb> <SEP> 4 <SEP> HTML <SEP> RESERVED <SEP> HTML
<tb> <SEP> 5-15-RESERVED
<tb>
High 4 Bit, Value: enumerated 0-15, compression method
EMI135.2

<tb> Value <SEP> Codec <SEP> Comment
<tb> <SEP> 0 <SEP> NONE <SEP> Uncompressed <SEP> 8 <SEP> bit <SEP> ASCII <SEP> codes
<tb> <SEP> TEXT7 <SEP> RESERVED-7 <SEP> Bit <SEP> Character <SEP> codes
<tb>
EMI136.1

2 <SEP> HUFF4 <SEP> RESERVED-4 <SEP> bit <SEP> Huffman <SEP> coded <SEP> ASCII
<tb> 3 <SEP> HUFF8 <SEP> RESERVED-8 <SEP> bit <SEP> Huffman <SEP> coded <SEP> ASCII
<tb> 4 <SEP> LZW <SEP> RESERVED-Lepel-Zev-Welsh <SEP> coded <SEP> ASCII
<tb> 5 <SEP> ARITH <SEP> RESERVED-Arithmetic <SEP> coded <SEP> ASCII
<tb> 6-15-RESERVED
<tb>
FontInfo-BYTE
Description-Size in low nibble (FontInfo & OxOF) Style in high nibble (FontInfo 4).

This field is ignored if the Type is WML or HTML.

Low 4 Bits Value: 0-15 FontSize
High 4 Bit Values: enumerated 0-15, FontSyle
Colour-WORD
Description-Textface colour
Valid Values:OxOOOO-OxEFFF, colour in 15 bit RGB(R5, G5, B5)
Ox8000-Ox80FF, colour as index into VideoData LUT (Ox80FF= transparent)
Ox8100-OxFFFF RESERVED
BackFill-WORD
Description-Background colour
Valid Values:OxOOOO-OxEFFF, colour in 15 bit RGB(R5, G5, B5)
Ox8000-Ox80FF, colour as index into VideoData LUT (Ox80FF = transparent)
Ox8100-OxFFFF RESERVED
Bounds-WORD
Description-Text boundary box (frame) in character units, Width in the LoByte (Bounds & OxOF) and height in the HiByte (Bounds > > 4). The text will be wrapped using the width and clipped for the height.

Valid Values: width = 1-255, height =1-255,
width 0-no wrapping performed,
height =0-no clipping performed
Xpos-WORD
Description-pos relative to object origin if defined else relative to 0,0 otherwise
Valid Values: OxOOOO-OxFFFF
Ypos-WORD
Description-pos relative to object origin if defined else relative to 0,0 otherwise
Valid Values: OxOOOO-OxFFFF
NOTE: Colours in the range ofOx80FO-Ox80FF are not valid colour indexes into VideoData LUTs since they only support up to 240 colours. Hence they are interpreted as per the following table. These colours should be mapped into the specific device/OS system colours as best possible according to the table. In the standard Palm OS UI only 8 colours are used and some of these colours are similar to the otherplatforms but not identical, this is indicated with anasterix. The missing 8 colours will have to be set by the application.

GrafDefinition
This packet contains the basic animation parameters. The actual graphic object definitions are contained in the GrafData packets, and the animation control in theobjControl packets.

Xpos-WORD
Description-XPos relative to object origin if defined relative to 0,0 otherwise
Valid Values:
Ypos-WORD
Description-XPos relative to object origin if defined relative to 0,0 otherwise
Valid Values:
FrameRate-WORD
Description-Frame delay in 8.8 fps
Valid Values:FrameSize-WORD
Description-Frame size in twips (1/20 pel)-used for scaling to fit scene space
Valid Values:
FrameCount-WORD
Description-How many frames in this animation
Valid Values:
Time-DWORD
Description-Time stamp in50ms resolution from start of scene
Valid Values:
VideoKey, VideoData, VideoTrp and AudioData
These packets contain codec specific compressed data. These packets contain codec specific compressed data.

Buffer sizes should be determined from the information conveyed in the VideoDefn and AudioDefn packets. Beyond the TypeTag VideoKey packets are similar to VideoData packets, differing only in their ability to encode transparency regions-VideoKey frames have no transparency regions. The distinction in type definition makeskeyframes visible at the file parsing level to facilitate browsing. VideoKey packets are an integral component of a sequence of VideoData packets; they are typically interspersed among them as part of the same packet sequence. VideoTrp packets represent frames that are non-essential to the video stream, thus they may be discarded by the Sky decoding engine
TextData
Textdata packets contain the ASCII character codes for text to be rendered. Whatever Serif system font are available one the client device should be used to render these fonts.

Serif fonts are to be used since proportional fonts require additional processing to render.

In the case where the specified Serif system font style is not available, then the closest matching available font should be used.

Plain text is rendered directly without any interpretation. White space characters other than LF (new line) characters and spaces and other special codes for tables and forms as specified below are totally ignored and skipped over. All text is clipped at scene boundaries.

The bounds box defines how text wrapping functions. The text will be wrapped using the width and clipped if it exceeds the height. If the bounds width is0 then no wrapping occurs. If the height is0 then no clipping occurs.

Table data is treated similarly as Plain text with the exception of LF that is used to denote end of rows and the CR character that is used to denote columns breaks.

WML and HTML is interpreted according to their respective standards, and the font style specified in this format is ignored. Images are not supported in WML and HTML.

To obtain streaming text data new TextData packets are sent to update the relevant object.

Also in normal text animation the rendering of TextData can be defined using ObjectControl packets.

GrafData
This packet contains all of the graphic shape and style definitions used for the graphics animation. This is a very simple animation data

type. Each shape is defined by a path, some attributes and a drawing style. One graphic object may be composed of an array of paths in any one GraphData packet. Animation of this graphic object can occur by clearing or replacing individual shape records array entires in the next frame, adding new records to the array can also be performed using the CLEAR and SKIP path types.

GraphData Packet
EMI140.1

<tb> Description <SEP> Type <SEP> Comment
<tb> NumShapes <SEP> BYTE <SEP> Number <SEP> of <SEP> shape <SEP> records <SEP> to <SEP> follow
<tb> Primitives <SEP> SHAPERecor <SEP> Array <SEP> of <SEP> Shape <SEP> Definitions
<tb> <SEP> d[]
<tb>
ShapeRecord
EMI140.2

<tb> Description <SEP> Type <SEP> Comment
<tb> Path <SEP> BYTE <SEP> Sets <SEP> the <SEP> path <SEP> of <SEP> the <SEP> shape <SEP> + <SEP> DELETE <SEP> operation
<tb> Style <SEP> BYTE <SEP> Defines <SEP> how <SEP> path <SEP> is <SEP> interpreted <SEP> and <SEP> rendered
<tb> Offset <SEP> IPOINT
<tb> Vertices <SEP> DPOINT <SEP> [] <SEP> Length <SEP> of <SEP> array <SEP> given <SEP> in <SEP> Path <SEP> low <SEP> nibble
<tb> FillColour <SEP> WORD <SEP> [] <SEP> Number <SEP> of <SEP> entries <SEP> depend <SEP> on <SEP> fill <SEP> style <SEP> and <SEP> # <SEP> vertices
<tb> LineColour <SEP> WORD <SEP> Optional <SEP> field <SEP> determined <SEP> by <SEP> style <SEP> field
<tb>
Path-BYTE
Description-Sets the path of the shape in the high nibble and the # vertices in low nibble
Low 4 Bits Value : 0-15 : number of vertices in poly paths
High 4 Bits Value: ENUMERATED: 0-15 defines the path shape
EMI140.3

<tb> <SEP> Value <SEP> Path <SEP> Comment
<tb> 0 <SEP> CLEAR <SEP> Deletes <SEP> SHAPERECORD <SEP> definition <SEP> from <SEP> array
<tb> 1 <SEP> SKIP <SEP> Skips <SEP> this <SEP> SHAPERECORD <SEP> in <SEP> the <SEP> array
<tb> 2 <SEP> RECT <SEP> Description-topleft <SEP> corner, <SEP> bottom <SEP> right <SEP> corner
<tb> <SEP> Valid <SEP> Values: <SEP> (0.. <SEP> 4096,0.. <SEP> 4096), <SEP> [0.. <SEP> 255,0.. <SEP> 255]...
<tb>
EMI141.1

<tb>

3 <SEP> POLY <SEP> Description-# <SEP> points, <SEP> initial <SEP> xy <SEP> value, <SEP> array <SEP> of <SEP> relative <SEP> pt <SEP> coords
<tb> <SEP> Valid <SEP> Values: <SEP> 0.. <SEP> 255, <SEP> (0.. <SEP> 4096,0.. <SEP> 4096), <SEP> [0.. <SEP> 255,
<tb> <SEP> 0.. <SEP> 255]...
<tb>

4 <SEP> ELLIPS <SEP> Description-centre <SEP> coord, <SEP> major <SEP> axis <SEP> radius, <SEP> minor
<tb> <SEP> E <SEP> axis <SEP> radius
<tb> <SEP> Valid <SEP> Values: <SEP> (0.. <SEP> 4096,0.. <SEP> 4096), <SEP> 0.. <SEP> 255,0.. <SEP> 255
<tb> 5-15 <SEP> RESERVED
<tb>
Style-BYTE
Description-Defines how path is interpreted
Low 4 Bits Value: 0-15 line thickness
High 4 Bits: BITFIELD: path rendering parameters. The default is not draw the shape at all so that it operates as an invisible hot region.

Bit[4] : CLOSED-If bit set then path is closed
Bit[5] : FILLFLAT-Default is no fill-if both fills then do nothing
Bit [6]: FILLSHADE-Default is no fill-if both fills then do nothing
Bit [7]: LINECOLOR-Default is no outline
UserControl
These are used to control the user-system and user-object interaction events. They are used as a back channel to return user interaction back to a server to effect server side control.

However if the file is not being streamed these user interactions are handled locally by the client. A number of actions can be defined for user-object control in each packet. The following actions are defined in this version. The user-object interactions need not be specified except to notify the server that one has occurred since the server knows what actions are valid.
EMI142.1

<tb>

<SEP> Keyboard <SEP> events <SEP> Play/Pause <SEP> system <SEP> control
<tb> <SEP> User, <SEP> system <SEP> i <SEP> feractons <SEP> i <SEP> I <SEP> II <SEP> !' <SEP> u <SEP> p'ej'i <SEP> bjl <SEP> m4ter <SEP> ? <SEP> Juti <SEP> $IIIV <SEP> ! <SEP> luf <SEP>
<tb> yttPen <SEP> events <SEP> (up, <SEP> down, <SEP> move, <SEP> dblclick) <SEP> Set <SEP> 2D <SEP> position, <SEP> visibility <SEP> (self, <SEP> other)
<tb> <SEP> Keyboard <SEP> events <SEP> Play/Pause <SEP> system <SEP> control
<tb> <SEP> Play <SEP> control <SEP> (play, <SEP> pause, <SEP> frame <SEP> advance, <SEP> Hyperlink-Goto <SEP> # <SEP> (Scene, <SEP> frame, <SEP> label,
<tb> <SEP> stop) <SEP> URL)
<tb> <SEP> Return <SEP> Form <SEP> Data <SEP> Hyperlink-Goto <SEP> next/prev, <SEP> (scene, <SEP> frame)
<tb> <SEP> Hyperlink-Replace <SEP> object <SEP> (self, <SEP> other)
<tb> <SEP> Hyperlink-Server <SEP> Defined
<tb>
The user-object interaction depends on what actions are defined for each object when they are clicked on by the user. The player may

know these actions through the medium of
ObjectControl messages. If it does not, then they are forwarded to an online server for processing. With user-object interaction the identification of the relevant object is indicated in the BaseHeaderobjid field. This applies to OBJCTRL and FORMDATA event types. For user-system interaction the value of theobjid field is 255. The Event type in UserControl packets specifies the interpretation of the key, HiWord and LoWord data fields.

Event-BYTE
Description-User Event Type
Valid Values: enumerated 0-255
EMI142.2

<tb> Value <SEP> Event <SEP> Type <SEP> Comment
<tb> 0PENDOWNUser <SEP> has <SEP> put <SEP> pen <SEP> down <SEP> on <SEP> touch <SEP> screen
<tb> 1 <SEP> PENUP <SEP> User <SEP> has <SEP> lifted <SEP> pen <SEP> up <SEP> from <SEP> touch <SEP> screen
<tb> 2 <SEP> PENMOVE <SEP> User <SEP> is <SEP> dragging <SEP> pen <SEP> across <SEP> touch <SEP> screen
<tb> 3 <SEP> PENDBLCLK <SEP> User <SEP> has <SEP> double <SEP> clicked <SEP> touch <SEP> screen <SEP> with <SEP> pen
<tb>
EMI143.1

4 <SEP> KEYDOWN <SEP> User <SEP> has <SEP> pressed <SEP> a <SEP> key
<tb> 5 <SEP> KEYUP <SEP> User <SEP> has <SEP> pressed <SEP> a <SEP> key
<tb> 6 <SEP> PLAYCTRL <SEP> User <SEP> has <SEP> activated <SEP> a <SEP> play/pause/stop <SEP> control
<tb> <SEP> button
<tb> 7 <SEP> OBJCTRL <SEP> User <SEP> has <SEP> clicked/activated <SEP> an <SEP> AV <SEP> object
<tb> 8 <SEP> FORMDATA <SEP> User <SEP> is <SEP> returning <SEP> form <SEP> data
<tb> 9-255-RESERVED
<tb> key, HiWord and LoWord-BYTE, WORD, WORD
Description-parameter data for different event types
Valid Values: The interpretation of these fields is as follows
EMI143.2

<tb> <SEP> Event. <SEP> HiWord <SEP> L6 <SEP> rd
<tb> PENDOWN <SEP> Key <SEP> code <SEP> if <SEP> key <SEP> held <SEP> down <SEP> X <SEP> position <SEP> Y <SEP> position
<tb> PENUP <SEP> Key <SEP> code <SEP> if <SEP> key <SEP> held <SEP> down <SEP> X <SEP> position <SEP> Y <SEP> position
<tb> PENMOVE <SEP> Key <SEP> code <SEP> if <SEP> key <SEP> held <SEP> down <SEP> X <SEP> position <SEP> Y <SEP> position
<tb> PENDBLCL <SEP> Key <SEP> code <SEP> if <SEP> key <SEP> held <SEP> down <SEP> X <SEP> position <SEP> Y <SEP> position
<tb> K
<tb> KEYDOWN <SEP> Key <SEP> code <SEP> Unicode <SEP> Key <SEP> code <SEP> 2""key <SEP> held <SEP> down
<tb> KEYUP <SEP> Key <SEP> code <SEP> Unicode <SEP> Key <SEP> code <SEP> 2nd <SEP> key <SEP> held <SEP> down
<tb> PLAYCTRL <SEP> Stop=0, <SEP> Start=I, <SEP> pause <SEP> = <SEP> 2 <SEP> RESERVED <SEP> RESERVED
<tb> OBJCTRL <SEP> Pen <SEP> Event <SEP> ID <SEP> Keycode <SEP> if <SEP> key <SEP> RESERVED
<tb> <SEP> held <SEP> down
<tb> FORMDATA <SEP> RESERVED <SEP> Length <SEP> of <SEP> data <SEP> field <SEP> RESERVED
<tb>
Time-WORD
Description-Time of user event = sequence number of activated object
Valid Values:O-OxFFFF
Data- (RESERVED-OPTIONAL)
Description-Text strings from form object
Valid Values: 0... 65535 bytes in length
Note: In the case of the PLAYCTRL events that pausing repeatedly when play is already paused should invoke a frame advance response from the server. Stopping should reset play to the start of the file/stream.

ObjectControl
ObjectControl packets are used to define the object-scene and system-scene interaction.

They also specifically define how objects are rendered and how scenes are played out. A new OBJCTRL packet is used for each frame to coordinate individual object layout. A number of actions can be defined for an object in each packet. The following actions are defined in this version
EMI144.1

<tb> <SEP> ~~
<tb> <SEP> w <SEP> Obect-system <SEP> actions'Syste,-sc <SEP> ne <SEP> aetons <SEP> .: <SEP> "
<tb> Set <SEP> 2D/3D <SEP> position <SEP> Goto <SEP> # <SEP> (Scene, <SEP> frame, <SEP> label, <SEP>
<tb> Set <SEP> 3D <SEP> Rotation <SEP> Goto <SEP> next, <SEP> previous, <SEP> (scene, <SEP> frame)
<tb> Set <SEP> scale/size <SEP> factor <SEP> Play/Pause
<tb> Set <SEP> visibility <SEP> Mute <SEP> audio
<tb> Set <SEP> label/title <SEP> (for <SEP> use <SEP> as <SEP> in <SEP> tool <SEP> tips) <SEP> IF <SEP> (scene, <SEP> frame, <SEP> object) <SEP> THEN <SEP> DO
<tb> <SEP> (action)
<tb> Set <SEP> background <SEP> colour <SEP> (nil
<tb> transparent)
<tb> Set <SEP> tweening <SEP> value <SEP> (for <SEP> animations)
<tb> Begin/end/duration/repeat <SEP> (loop)
<tb>
EMI145.1

<tb> impliit
<tb>
ControlMask-BYTE
o Description-Bit field-The control mask defines controls common to
Object level and System level operations. Following the ControlMask is an
optional parameter indicating the object id of the affected object. If there is

no affected object ID specified then the affected object id is the object id of the base header. The type of ActionMask (object or system scope) following the ControlMask is determined by the affected object id.

'Bit : [7] CONDITION-What is needed to perform these actions
'Bit : [6] BACKCOLR-Set colour of object background
'Bit : [5] PROTECT-limit user modification of scene objects
'Bit : [4] JUMPTO-replace the source stream for an object with another
-Bit : [3] HYPERLINK-sets hyperlink target
-Bit : [2] OTHER-object id of the affected object will follow(255=system)
'Bit :[1] SETTIMER-Set a timer and start counting down
'Bit :[0] EXTEND-RESERVED for future expansionControlObject-BYTE (Optional)
o Description: Object ID of affected object. Is included if bit 2 of ControlMask is set.
o Valid values: 0-255
Timer-WORD (Optional)
o Description: Topnibble=timer number, bottom 12 bits = time setting
o Top nibble, valid values: 0-15 timer number for this object.
o Bottom 12 bits valid range: 0-4096 time setting in100ms steps
ActionMask [OBJECT scope]-WORD
o Description-Bit field-This defines what actions are specified in this record and the parameters to follow. There are two versions of this one for object the other for system scope. This field defines actions that apply to media objects.
o Valid Values: For objects each one of the 16 bits in the ActionMask identifies an action to be taken. If a bit is set, then additional associated parameter values follow this field.

'Bit : [15] BEHAVIOR-indicates that this action and conditions remain with the object even after the actions have been executed
'Bit : [14] ANIMATE-multiple control points defining path will follow
'Bit : [13] MOVETO-set screen position
-Bit : [12] ZORDER-set depth
-Bit :[11] ROTATE-3D Orientation
-Bit : [10] ALPHA-Transparency
-Bit : [9] SCALE-Scale/size
-Bit : [8] VOLUME-set loundness
-Bit : [7] FORECOLR-set/change foreground colour
-Bit : [6] CTRLLOOP-repeat the next # actions (if set else ENDLOOP)
'Bit : [5] ENDLOOP-if loopingcontrol/animation then break it
'Bit : [4] BUTTON-define penDown image for button
'Bit : [3]COPYFRAME-copies the frame from object into this object (checkbox)
-Bit : [2]CLEAR WAITINGACTIONS-clears waiting actions
-Bit :[1] OBJECT MAPPING-specifies the object mapping between streams
-Bit :[0] ACTIONEXTEND-Extended Action Mask follows ActionExtend [OBJECT scope]-WORD
o Description-Bit field-RESERVED
ActionMask [SYSTEM scope]-BYTE
o Description-Bit field-This defines what actions are specified in this record and the parameters to follow. There are two versions of this one for object the other for system scope. This field defines actions that have scene wide scope.
o Valid Values: For systems each one of the 16 bits in the ActionMask identifies an action to be taken. If a bit is set then additional associated parameter values follow this field
'Bit : [7] PAUSEPLAY-if playing pause indefinitively
'Bit : [6] SNDMUTE-if sounding then mute if muted then sound
'Bit :[5] SETFLAG-Sets user assignable system flag value
'Bit : [4] MAKECALL-change/open the physical channel
Bits : [3] SENDDTMF-Send DTMF tones on voice call
-Bits : [2-0]-RESERVEDParams-BYTE array
o Description-Byte array. Most of the actions defined in the above bit fields use additional parameters. The parameters used as indicated by the bit field value being set are specified here in the same order as the bit field from top (15) to bottom(0) and order of masks, ActionMask then [Object/System] Mask (except for the affected object id which has already been specified between the two). These parameters may include optional fields, these are shown as yellow rows in the tables below.
o CONDITION bit-Consists of one or more state records chained together, each record can also have an optional frame number field after it. The conditions within each record are logically ANDed together. For greater flexibility additional records can be chained through bit0 to create logical OR conditions. In addition to this, multiple, distinct definition records may exist for any one object creating multiple conditional control paths for each object.
EMI148.1

<tb>

Param <SEP> Type <SEP> Comment
<tb> State <SEP> WORD <SEP> What <SEP> is <SEP> needed <SEP> to <SEP> perform <SEP> these <SEP> actions, <SEP> bit-field
<tb> <SEP> (logically <SEP> ANDed)
<tb> <SEP> * <SEP> Bit: <SEP> [15] <SEP> playing//continuous <SEP> playing
<tb> <SEP> * <SEP> Bit: <SEP> [14] <SEP> paused <SEP> 11 <SEP> playing <SEP> is <SEP> paused

<tb> <SEP> 'Bit <SEP> : <SEP> [13] <SEP> stream <SEP> 11 <SEP> streaming <SEP> from <SEP> remote <SEP> server
<tb> <SEP> * <SEP> Bit: <SEP> [12] <SEP> stored <SEP> 11 <SEP> playing <SEP> from <SEP> local <SEP> storage
<tb> <SEP> # <SEP> Bit: <SEP> [11] <SEP> buffered//is <SEP> object <SEP> frame <SEP> # <SEP> buffered? <SEP> (true <SEP> if
<tb> <SEP> stored)
<tb> <SEP> # <SEP> Bit: <SEP> [10] <SEP> overlap//what <SEP> object <SEP> do <SEP> we <SEP> need <SEP> to <SEP> be
<tb> <SEP> dropped <SEP> on?
<tb> <SEP> # <SEP> Bit: <SEP> [9] <SEP> event//what <SEP> user <SEP> event <SEP> needs <SEP> to <SEP> be
<tb> <SEP> happening
<tb> <SEP> # <SEP> Bit: <SEP> [8] <SEP> wait <SEP> // <SEP> do <SEP> we <SEP> wait <SEP> for <SEP> conditions
<tb> <SEP> to <SEP> become <SEP> true
<tb> <SEP> # <SEP> Bit: <SEP> [7] <SEP> userflags//tests <SEP> user <SEP> flags <SEP> which <SEP> follow
<tb> <SEP> 'Bit <SEP> : <SEP> [6] <SEP> TimeUp//Timer <SEP> has <SEP> expired
<tb> <SEP> Bit: <SEP> [5-1] <SEP> RESERVED
<tb> <SEP> 'Bit <SEP> : <SEP> [0] <SEP> OrState//OrState <SEP> condition <SEP> record <SEP> follow
<tb> Frame <SEP> WORD <SEP> (optional) <SEP> frame <SEP> number <SEP> for <SEP> bit <SEP> 11 <SEP> condition
<tb> Object <SEP> BYTE <SEP> (optional) <SEP> object <SEP> ID <SEP> for <SEP> bit <SEP> 10 <SEP> condition, <SEP> invisible <SEP> objects
<tb> <SEP> can <SEP> be <SEP> used
<tb> Event <SEP> WORD <SEP> High <SEP> BYTE: <SEP> the <SEP> event <SEP> field <SEP> from <SEP> the <SEP> UserControl <SEP> Packet
<tb> <SEP> Low <SEP> BYTE: <SEP> the <SEP> key <SEP> field <SEP> from <SEP> the <SEP> UserControl <SEP> Packet,
<tb> <SEP> OxFF <SEP> ignore <SEP> keys, <SEP> 0x00 <SEP> no <SEP> key <SEP> being <SEP> pressed
<tb> User <SEP> DWOR <SEP> High <SEP> WORD: <SEP> mask <SEP> indicating <SEP> which <SEP> flags <SEP> to <SEP> check
<tb>
EMI149.1

flags <SEP> D <SEP> Low <SEP> WORD: <SEP> mask <SEP> indicating <SEP> the <SEP> values <SEP> of <SEP> user <SEP> flags <SEP> (set
<tb> <SEP> or <SEP> not <SEP> set)
<tb> TimeU <SEP> BYTE <SEP> High <SEP> nibble: <SEP> RESERVED
<tb> p <SEP> Low <SEP> nibble: <SEP> timer <SEP> id <SEP> number <SEP> (0-15)
<tb> State <SEP> WORD <SEP> Same <SEP> bit <SEP> field <SEP> as <SEP> the <SEP> previous <SEP> state <SEP> field, <SEP> but <SEP> is <SEP> logically
<tb> <SEP> ORed <SEP> with <SEP> it
<tb> <SEP> ... <SEP> WORD...
<tb> o ANIMATE bit set-If the animate bit is set then animation parameters follow specifying the times and interpolation of the animation. The animate bit also affects the number of MOVETO, ZORDER, ROTATE, ALPHA, SCALE, and VOLUME parameters that exist in this control. Multiple values will occur for each parameter, one value for each control point.
EMI149.2

<tb>

<SEP> Param <SEP> Type <SEP> Commet
<tb> AnimCtr <SEP> BYTE <SEP> High <SEP> nibble <SEP> : <SEP> Number <SEP> of <SEP> control <SEP> points-1
<tb> 1 <SEP> Low <SEP> nibble: <SEP> path <SEP> control
<tb> <SEP> * <SEP> Bit <SEP> [3]: <SEP> Looping <SEP> Animation
<tb> <SEP> * <SEP> Bit <SEP> [2]: <SEP> RESERVED
<tb> <SEP> * <SEP> Bits <SEP> [1.. <SEP> 0] <SEP> : <SEP> enum, <SEP> Path <SEP> type- <SEP> {0: <SEP> linear, <SEP> 1: <SEP> Quadratic,
<tb> <SEP> 2: <SEP> Cubic}
<tb> Start <SEP> WORD <SEP> Start <SEP> time <SEP> of <SEP> animation, <SEP> from <SEP> scene <SEP> start <SEP> or <SEP> condition <SEP> in
<tb> time <SEP> 50ms <SEP> steps
<tb> Duration <SEP> WORD <SEP> [Array <SEP> of <SEP> durations <SEP> in <SEP> 50ms <SEP> increments, <SEP> length <SEP> = <SEP> control
<tb> s] <SEP> points-1
<tb> o MOVETO bit set
EMI149.3

<tb> Param <SEP> Type <SEP> Comment
<tb> Xpos <SEP> WOR <SEP> X <SEP> position <SEP> to <SEP> move <SEP> to, <SEP> relative <SEP> to <SEP> current <SEP> pos
<tb> <SEP> D
<tb> <SEP> Ypos <SEP> WOR <SEP> Y <SEP> position <SEP> to <SEP> move <SEP> to, <SEP> relative <SEP> to <SEP> current <SEP> pos
<tb> <SEP> D
<tb> o ZORDER bit set
EMI150.1

<SEP> Comment
<tb> Depth <SEP> WOR <SEP> increases <SEP> away <SEP> from <SEP> viewer, <SEP> values <SEP> of <SEP> 0,256,512,768
<tb> <SEP> D <SEP> etc <SEP> reserved
<tb> o ROTATE bit set
EMI150.2

<tb> Param <SEP> Type <SEP> Comment
<tb> Xrot <SEP> BYTE <SEP> X <SEP> axis <SEP> rotation, <SEP> absolute <SEP> in <SEP> degress* <SEP> 255/360,
<tb> Yrot <SEP> BYTE <SEP> Y <SEP> axis <SEP> rotation, <SEP> absolute <SEP> in <SEP> degrees <SEP> * <SEP> 255/360
<tb> Zrot <SEP> BYTE <SEP> Z <SEP> axis <SEP> rotation, <SEP> absolute <SEP> in <SEP> degrees <SEP> * <SEP> 255/360
<tb> o ALPHA bit set o SCALE bit set
EMI150.3

<tb> Param <SEP> Type <SEP> Comment

<tb> scale <SEP> WOR <SEP> Size/Scale <SEP> in <SEP> 8.8 <SEP> fixed <SEP> int <SEP> format
<tb> <SEP> D
<tb> o VOLUME bit set o BACKCOLR bit set
EMI150.4

Param <SEP> Type <SEP> Comment
<tb> fillcolr <SEP> WOR <SEP> Same <SEP> format <SEP> as <SEP> SceneDefinition <SEP> Backcolor <SEP> (nil <SEP> =
<tb> <SEP> D <SEP> transparent)
<tb> o PROTECT bit set
EMI151.1

disabled
<tb> Bit <SEP> : <SEP> [7] <SEP> move//prohibit <SEP> moving <SEP> objects
<tb> Bit <SEP> : <SEP> [6] <SEP> alpha//prohibit <SEP> changing <SEP> alpha <SEP> value
<tb> Bit <SEP> : <SEP> [5] <SEP> depth//prohibit <SEP> changing <SEP> depth <SEP> value
<tb> . <SEP> Bit <SEP> : <SEP> [4] <SEP> clicks//disable <SEP> click <SEP> through <SEP> behaviour
<tb> Bit <SEP> : <SEP> [3] <SEP> drag//disable <SEP> dragging <SEP> of <SEP> objects
<tb> *Bit <SEP> : <SEP> [2.. <SEP> 0]//RESERVED
<tb> o CTRLLOOP bit set
EMI151.2

<tb> Param <SEP> Type <SEP> Comment
<tb> <SEP> Repeat <SEP> BYTE <SEP> Repeat <SEP> the <SEP> next <SEP> # <SEP> actions <SEP> for <SEP> this <SEP> object-clicking <SEP> on <SEP> object
<tb> <SEP> to <SEP> break <SEP> loop
<tb> o SETFLAG bit set
EMI151.3

<tb> Param <SEP> Type <SEP> Comment
<tb> <SEP> Flag <SEP> BYTE <SEP> Top <SEP> nibble <SEP> = <SEP> flag <SEP> number, <SEP> bottom <SEP> nibble <SEP> if <SEP> true <SEP> set <SEP> flag <SEP> else
<tb> <SEP> reset <SEP> flag,
<tb> o HYPERLINK bit set
EMI151.4

<SEP> Type <SEP> Comment
<tb> hLink <SEP> BYTE <SEP> Sets <SEP> hyperlink <SEP> target <SEP> URL <SEP> for <SEP> click <SEP> through
<tb> o JUMPTO bit set
EMI151.5

Param <SEP> Type <SEP> Comment
<tb> <SEP> scene <SEP> BYTE <SEP> Goto <SEP> scene <SEP> # <SEP> if <SEP> value=0xFF <SEP> goto <SEP> hyperlink <SEP> (250=library)
<tb> <SEP> stream <SEP> BYTE <SEP> [optional] <SEP> Stream <SEP> # <SEP> if <SEP> value <SEP> = <SEP> 0 <SEP> then <SEP> read <SEP> optional <SEP> object <SEP> id
<tb> <SEP> object <SEP> BYTE <SEP> [optional] <SEP> object <SEP> id <SEP> #
<tb> o BUTTON bit set
EMI152.1

stream <SEP> BYTE <SEP> Stream <SEP> # <SEP> if <SEP> value <SEP> = <SEP> 0 <SEP> then <SEP> read <SEP> optional <SEP> object <SEP> id
<tb> object <SEP> BYTE <SEP> [optional] <SEP> object <SEP> id <SEP> #
<tb> o COPYFRAME bit set o OBJECTMAPPING bit set-when an object jumps to another stream the stream may use different object ids to the current scene. Hence an object mapping is specified in the same packet containing a JUMPTO command.
EMI152.2

<tb>

<SEP> Param <SEP> Type <SEP> Comment
<tb> Objects <SEP> BYTE <SEP> Number <SEP> of <SEP> objects <SEP> to <SEP> be <SEP> mapped
<tb> Mappin <SEP> WORD <SEP> [Array <SEP> of <SEP> words, <SEP> length <SEP> = <SEP> objects
<tb> g] <SEP> High <SEP> BYTE: <SEP> object <SEP> id <SEP> being <SEP> used <SEP> in <SEP> the <SEP> stream <SEP> we <SEP> are
<tb> <SEP> jumping <SEP> to
<tb> <SEP> Low <SEP> BYTE: <SEP> object <SEP> id <SEP> of <SEP> the <SEP> current <SEP> scene <SEP> which <SEP> the <SEP> new
<tb> <SEP> object <SEP> ids <SEP> will <SEP> be <SEP> mapped <SEP> to.
<tb> o MAKECALL bit set
EMI152.3

<tb> <SEP> Comment
<tb> channe <SEP> DWORD <SEP> Phone <SEP> number <SEP> of <SEP> new <SEP> channel
<tb> 1
<tb> o SENDDTMF bit set
EMI152.4

<tb> Param <SEP> Type <SEP> Comment
<tb> <SEP> DTMF <SEP> BYTE <SEP> [] <SEP> DTMF <SEP> string <SEP> to <SEP> be <SEP> sent <SEP> on <SEP> channel
<tb>
Notes:
w There are no parameters for the PAUSEPLAY and SNDMUTE actions as these are binary flags.

Button states can be created by having an extra image object that is set to be initially transparent.

When the user clicks down on the button object, this is then replaced with the invisible object that is set to visible using the button behaviour field and reverts to the original state when the pen is lifted.

ObjLibControl
ObjLibCtrl packets are used to control the persistent local object library that the player maintains. In one sense the local object library may be considered to store resources. A total of 200 user objects and 55 system objects can be stored in each library. During playback the object library can be directly addressed by using objectid = 250 for the scene. The object library is very powerful and unlike the font library supports both persistence and automatic garbarge collection..

The Objects are inserted into the object library through a combination of ObjLibCtrl packets and SceneDefn packets which have the ObjLibrary bit set in the Mode bit field [bit0]. Setting this bit in the SceneDefn packet tells the player that the data to follow is not to be played out directly but is to be used to populate the object library. The actual object data for the library is not packaged in any special manner it still consists of definition packets and data packets. The difference is that there is now an associated ObjLibCtrl packet for each object that instructs the player what to do with the object data in the scene.

Each ObjLibCtrl packet contains management information for the object with the sameobjid in the base header. A special case of ObjLibCtrl packets are those that have objectid in the base header set to 250. These are used to convey library system management commands to the player.

The present invention described herein may be conveniently implemented using a conventional general purpose digital computer or microprocessor programme according to the teachings of the present specification, as will be apparent to those skilled in the computer art. Appropriate software coding can readily be prepared by skilledprogrammers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art. The invention may also be implemented by the preparation of application specific integrated circuits or by interconnecting an appropriate network of conventional component circuits, as will be readily apparent to those skilled in the art. It is to be noted that this invention not only includes the encoding processes and systems disclosed herein, but also includes corresponding decoding systems and processes which may be implemented to operate to decode the encoded bit streams or files generated by the encoders in basically the opposite order of encoding, eluding certain encoding specific steps.

The present invention includes a computer program product or article of manufacture which is a storage medium including instructions which can be used to program a computer or computerized device to perform a process of the invention. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, CD-ROMs, and magneto-optical disks,ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions.

The invention also includes the data or signal generated by the encoding process of the invention. This data or signal may be in the form of an electromagnetic wave or stored in a suitable storage medium.

Many modifications will be apparent to those skilled in the art without departing from the spirit and scope of the present invention as herein described

AN OBJECT ORIENTED VIDEO SYSTEM

Claims of WO0131497

CLAIMS:1. A method of generating an object oriented interactive multimedia file, including:
encoding data comprising at least one of video, text, audio, music and/or graphics elements as a video packet stream, text packet stream, audio packet stream, music packet stream and/or graphics packet stream respectively;
combining said packet streams into a single self-contained object, said object containing its own control information;
placing a plurality of said objects in a data stream; and
grouping one or more of said data streams in a single contiguous self-contained scene, said scene including format definition as the initial packet in a sequence of packets.

2. A method of generating an interactive multimedia file according to claim 1, including combining one or more of said scenes.

3. A method of generating an interactive multimedia file according to claim 1 wherein a single scene contains an object library.

4. A method of generating an interactive multimedia file according to claim 1 wherein data for configuring customisable decompression transforms is included within said objects.

5. A method of generating an interactive object oriented multimedia file according to claim 1 wherein object control data is attached to objects which are interleaved into a video bit stream, and said object control data controls interaction behaviour, rendering parameters, composition, and interpretation of compressed data.

6. A methodof generating an interactive object oriented multimedia file according to claim 1 comprising a hierarchical directory structure wherein first level directory data comprising scene information is included with the first said scene, second level directory data comprising stream information is included with one or more of said scenes, and wherein third level directory data comprising information identifying the location of intraframes is included in said data stream.

7. A method of generating an object oriented interactive multimedia file, including:
encoding data comprising at least one of video and audio elements as a video packet stream and audio packet stream respectively;
combining said packet streams into a single self-contained object;
placing said object in a data stream;
placing said stream in a single contiguous self-contained scene, said scene including format definition; and
combining a plurality of said scenes.

8. A method of generating an interactive object oriented multimedia file according to claim 1, wherein said object control data takes the form of messages encapsulated within object control packets and represents parameters for rendering video and graphics objects, for defining the interactive behaviour of said objects, for creating hyperlinks to and from said objects, for defining animation paths for said objects, for defining dynamic media composition parameters, for assigning values to user variables, for redirecting or retargeting the consequences of interactions with objects and other controls from one object to another, for attaching executable behaviours to objects, including voice calls and starting and stop timers, and for defining conditions for the execution of control actions.

9. A method of generating an interactive object oriented multimedia file according to claim 7, wherein said rendering parameters represent object transparency, scale, volume, position, z-order, background colour and rotation, where said animation paths affect any of said rendering parameters, said hyperlinks support non-linear video and links to other video files, individual scenes within a file, and other object streams within a scene as targets, said interactive behaviour data includes the pausing of play and looping play, returning user information back to the server, activating or deactivating object animations, defining menus, and simple forms that can register user selections.

10. A method of generating an interactive object oriented multimedia file according to claim 7, wherein conditional execution of rendering actions or object behaviours is provided and conditions take the form of timer events, user events, system events, interaction events, relationships between objects, user variables, and system status such as playing, pausing, streaming or stand-alone play.

11. A method of mapping in real time from a non-stationary three-dimensional data set onto a single dimension, comprising the stepsof :
pre-computing said data; encoding said mapping;
transmitting the encoded mapping to a client; and
said client applying said mapping to the said data.

12. A method of mapping in real time from a non-stationary three-dimensional data set onto a single dimension according to claim 11, wherein said data set comprises a colour video frame and said pre-computing comprises a vector quantisation process;
determining the closest codebook vector for each cell in the mapping process;
performing said encoding using an octree representation;
sending said encoded octree to a decoder; and
said decoder then applying mapping to said data set.

13. An interactive multimedia file format comprising single objects containing video, text, audio, music, and/or graphical data wherein at least one of said objects comprises a data stream, and at least one of said data streams comprises a scene, at least one of said scenes comprises a file, and wherein directory data and metadata provide file information.

14. A system for dynamically changing the actual content of a displayed video in an object-oriented interactive video system comprising:
a dynamic media composition process including an interactive multimedia file format including objects containing video, text, audio, music, and/or graphical data wherein at least one of said objects comprises a data stream, at least one of said data streams comprises a scene, at least one of said scenes comprises a file;
a directory data structure for providing file information;
selecting mechanism for allowing the correct combination of objects to be composited together;
a data stream manager for using directory information and knowing the location of said objects based on said directory information; and
control mechanism for inserting, deleting, or replacing in real time while being viewed by a user, said objects in said scene and said scenes in said video.

15. A system according to claim 14 including remote server non-sequential access capability, selection mechanism for selecting appropriate data components from each object stream, interleaving mechanism for placing said data components into a final composite data stream, and wireless transmission mechanism for sending said final composite stream to a client.

16. A system according to claim 14 including remote server non-sequential access capability, including a mechanism for executing library management instructions delivered to said system from said remote server, said server capable of querying said library and receiving information about specific objects contained therein, and inserting, updating, or deleting the contents of said library; and said dynamic media composition engine capable of sourcing object data stream simultaneously both from said library and remote server if required.

17. A system according to claim 14 including a local server providing offline play mode;
a storage mechanism for storing appropriate data components in local files;
selection mechanism for selecting appropriate data components from separate sources;
a local data file including multiple streams for each scene stored contiguously within said file;
access mechanism for said local server to randomly access each stream within a said scene;
selection mechanism for selecting said objects for rendering;
a persistent object library for use in dynamic media composition capable of being managed from said remote server, said objects capable of being stored in said library with full digital rights management information;
software available to a client for executing library management instructions delivered to it from said remote server, said server capable of querying said library and receiving information about specific objects contained therein, and inserting, updating, or deleting the contents of said library; and
said dynamic media composition engine capable of sourcing object data stream simultaneously both from said library and remote server.

18. A system according to claim 14, wherein each said stream includes an end of stream packet for demarcating stream boundaries, said first stream in a said scene containing descriptions of said objects within said scene;
object control packets within said scene provide information for interactivity, changing the source data for a particular object to a different stream;
reading mechanism in said server for reading more than one stream simultaneously from within a said file when performing local playback; and
mechanism for managing an array or linked list of streams, data stream manager capable of reading one packet from each stream in a cyclical manner; storage mechanism for storing the current position in said file; and storage mechanism for storing a list of referencing objects.

19. A system according to claim 14, wherein data is streamed to a media player client, said client capable of decoding packets received from the remote server and sending back user operations to said server, said server responding to user operations such as clicking, and modifying said data sent to said client, each said scene containing a single multiplexed stream composed of one or more objects, said server capable of composing scenes in realtime by multiplexing multiple object data streams based on client requests to construct a single multiplexed stream for any given scene, and wireless streaming to said client for playback.

20. A system according to claim 14 including playing mechanism for playing a plurality of video objects simultaneously, each of said video objects capable of originating from a different source, said server capable of opening each of said sources, interleaving the bit streams, adding appropriate control information and forwarding the new composite stream to said client.

21. A system according to claim 14 including a data source manager capable of randomly accessing said source file, reading the correct data and control packets from said streams which are needed to compose the display scene, and including a server multiplexer capable of receiving input from multiple source manager instances with single inputs and from said dynamic media composition engine, said multiplexer capable of multiplexing together object data packets from said sources and inserting additional control packets into said data stream for controlling the rendering of component objects in the composite scene.

22 A system according to claims 14 including an XML parser to enable programmable control of said dynamic media composition through IAVML scripting.

23. A system according to claims 14, wherein said remote server is capable of accepting a number of inputs from the server operator to further control and customize said dynamic media composition process, said inputs including user profile, demographics, geographic location, or the time of day.

24. A system according to claims 14, wherein said remote server is capable of accepting a number of inputs from the server operator to further control and customize said dynamic media composition process, said inputs including a log of user interaction such as knowledge of what advertisements have success with a user.

25. An object oriented interactive multimedia file, comprising:
a combination of one or more of contiguous self-contained scenes;
each said scene comprising scene format definition as the first packet, and a group of one or more data streams following said first packet;
each said data stream apart from first data stream containing objects which may be optionally decoded and displayed according to a dynamic media composition process as specified by object control information in said first data stream; and
each said data stream including one or more single self-contained objects and demarcated by an end stream marker; said objects each containing it's own control information and formed by combining packet streams; said packet streams formed by encoding raw interactive multimedia data including at least one or a combination of video, text, audio, music, or graphics elements as a video packet stream, text packet stream, audio packet stream, music packet stream and graphics packet stream respectively.

26. An object-oriented interactive video system including an interactive multimedia file format according to claim 25 including:
server software for performing said dynamic media composition process, said process allowing the actual content of a displayed video scene to be changed dynamically in real-time while a user views said video scene, and for inserting, replacing, or adding any of said scene's arbitrary shapedvisual/audio video objects; and
a control mechanism to replace in-picture objects by other objects to add or delete in-picture objects to or from a current scene to perform saidprocess in a fixed, adaptive, or user-mediated mode.

27. An object oriented interactive multimedia file according to claim 25 including data for configuring customisable decompression transforms within said scenes.

28. An object-oriented interactive video system including an interactive multimedia file format according to claim 25 including:
a control mechanism to provide a local object library to support said process, said library including a storage means for storing objects for use in said process, control mechanism to enable management of said library from a streaming server, control mechanism for providing versioning control for said library objects, and for enabling automatic expiration of non persistent library objects; and
control mechanism for updating objects automatically from said server, for providing multilevel access control for said library objects, and for supporting a unique identity, history and status for each of said library objects.

29. An object-oriented interactive video system including an interactive multimedia file format according to claim 25 including:
a control mechanism for responding to a user click on a said object in a session by immediately performing said dynamic media composition process; and
control mechanism for registering a user for offline follow-up actions, and for moving to a new hyperlink destination at the end of said session.

30. A method of real-time streaming of file data in the object oriented file format according to claim 25, over a wireless network whereby a scene includes only one stream, and said dynamic media composition engine interleaves objects from other streams at an appropriate rate into the said first stream.

31. A method of real-time streaming of file data in the object oriented file format according to claim 25, over a wireless network whereby a scene includes only one stream, and said dynamic media composition engine interleaves objects from other streams at an appropriate

rate into the said first stream.

32. A method according to claim 30 of streaming live video content to a user where said other streams include streams which are encoded in real time.

33. A method according to claim 31 of streaming live video content to a user comprising the following steps:
said user connecting to a remote server; and
said user selecting a camera location to view within a region handled by the operator/exchange; 34. A method according to claim 31 of streaming live video content to a user comprising the following steps:
said user connects to a remote server; and
said user's geographic location, derived from a global positioning system or cell triangulation, is used to automatically provide a selection of camera locations to view for assistance with said user's selection of a destination.

35. A method according to claim 31 of streaming live traffic video content to a user comprising the following steps:
said user registers for a special service where a service provider calls said user and automatically streams video showing a motorist's route that may have a potential problem area;
upon registering said user may elect to nominate a route for this purpose, and may assist with determining said route; and
said system tracks said user's speed and location to determine the direction of travel and route being followed, said system could then search its list of monitored traffic cameras along potential routes to determine if any sites are problem areas, and if any problems exist, said system notifies said user and plays a video to present the traffic information and situation.

36. A method of advertising according to claim 26, wherein said dynamic media composition process selects objects based on a subscriber's own profile information, stored in a subscriber profile database.

37. A method of providing a voice command operation of a low power device capable of operating in a streaming video system, comprising the following steps:
capturing a user's speech on said device;
compressing said speech;
inserting encoded samples of said compressed speech into user control packets;
sending said compressed speech to a server capable of processing voice commands;
said server performs automatic speech recognition;
said server maps the transcribed speech to a command set;
said system checks whether said command is generated by said user or said server;
if said transcribed command is from said server, said server executes said command;
if said transcribed command is from said user said system forwards said command to said user device; and
said user executes said command.

38. A method of providing a voice command operation of a low power device capable of operating in a streaming video system, according to claim 37, wherein:
said system determines whether transcribed command is pre-defined;
if said transcribed command is not pre-defined, said system sends said transcribed text string to said user; and
said user inserts said text string into an appropriate text field.

39. An image processing method, comprising the stepof :
generating a colour map based on colours of an image;
determining a representation of the image using the colour map; and
determining a relative motion of at least a section of the image which is represented using the colour map.

40. A method according to claim 39, further comprising the step of encoding the representation of the image.

41. A method according to claim 39, further comprising the step of encoding the relative motion.

42. A method according to claim 39, further comprising the step of encoding the representation of the image and the relative motion.

43. A method according to claim 39, wherein said generating step comprises performing a colour quantisation in order to generate the colour map.

44. A method according to claim 43, wherein said generating step further comprises creating the colour map based on a previously determined colour map of a proximate frame.

45. A method according to claim 44, wherein said creating step comprises reorganising the colour map based on the previously determined colour map so that colours of pixels from the proximate frame which are carried over to a current frame are mapped to same indexes of the colour map.

46. A method according to claim 44, wherein said creating step comprises correlating the colour map to the previously determined colour map.

47. A method according to claim 39, wherein said step of determining a relative motion comprises determining a motion vector for the at least a section of the image.

48. An image processing method, comprising creating a quadtree for encoding a representation of an image.

49. A method according to claim 48, wherein the encoding step comprises creating the quadtree to have a transparent leaf representation.

50. A method according to claim 49, wherein the encoding step comprises creating the quadtree to have the transparent leaf representation which is utilized to represent arbitrary shaped objects.

51. A method according to claim 50, wherein the encoding step comprises creating the quadtree to have bottom level node type elimination.

52. A method of determining an encoded representation of an image comprising:
analyzing a number of bits utilized to represent a colour;
representing the colour utilizing a first flag value and a first predetermined number of bits, when the number of bits utilized to represent the colour exceeds a first value; and
representing the colour utilizing a second flag value and a second predetermined number of bits, when the number of bits utilized to represent the colour does not exceed a first value.

53. A method according to claim 52, wherein the step of representing the colour utilizing the first flag value comprises representing the colour using the first predetermined number of bits which is eight; and
the step of representing the colour utilizing the second flag value comprises representing the colour using the second predetermined number of bits which is four.

54. An image processing system, comprising means for generating a colour map based on colours of an image;
means for determining a representation of the image using the colour map; and
means for determining a relative motion of at least a section of the image which is represented using the colour map.

55. A system according to claim 54, further comprising means for encoding the representation of the image.

56. A system according to claim 54, further comprising means for encoding the relative motion.

57. A system according to claim 54, further comprising means for encoding the representation of the image and the relative motion.

58. A system according to claim 54, wherein said means for generating comprises means for performing a colour quantisation in order to generate the colour map.

59. A system according to claim 58, wherein said means for generating further comprises means for creating the colour map based on a previously determined colour map of a proximate frame.

60. A system according to claim 59, wherein said means for creating comprises means for reorganizing the colour map based on the previously determined color map so that colours of pixels from the proximate frame which are carried over to a current frame are mapped to same indexes of the colour map.

61. A system according to claim 59, wherein said means for creating comprises means for correlating the colour map to the previously determined colour map.

62. A system according to claim 54, wherein said means for determining a relative motion comprises means for determining a motion vector for the at least a section of the image.

63. An image encoding system comprising means for creating a quadtree for encoding a representation of an image.

64. A system according to claim 63, wherein the means for encoding comprises means for creating the quadtree to have a transparent leaf representation.

65. A system according to claim 64, wherein the means for encoding comprises means for creating the quadtree to have the transparent leaf representation which is utilized to represent arbitrary shaped objects.

66. A system according to claim 65, wherein the means for encoding comprises means for creating the quadtree to have bottom level node type elimination.

67. An image encoding system for determining an encoded representation of an image comprising :
means for analyzing a number of bits utilized to represent a colour;
means for representing the colour utilizing a first flag value and a first predetermined number of bits, when the number of bits utilized to represent the colour exceeds a first value; and
means for representing the colour utilizing a second flag value and a second predetermined number of bits, when the number of bits utilized to represent the colour does not exceed a first value.

68. A system according to claim 67, wherein the means for representing the color utilizing the first flag value comprises representing the color using the first predetermined number of bits which is eight; and
the step of representing the color utilizing the second flag value comprises representing the color using the second predetermined number of bits which is four.

69. A method of processing objects, comprising the stepsof :
parsing information in a script language;
reading a plurality of data sources containing a plurality of objects in the form of at least one of video, graphics, animation, and audio;
attaching control information to the plurality of objects based on the information in the script language; and
interleaving the plurality of objects into at least one of a data stream and a file.

70. A method according to claim 69, further comprising the step of inputting information from a user, wherein the step of attaching is performed based on the information in the script language and the information from the user.

71. A method according to claim 69, further comprising the step of inputting control information selected from at least one of profile information, demographic information, geographic information, and temporal information, wherein the step of attaching is performed based on the information in the script language and the control information.

72. A method according to claim 71, further comprising the step of inputting information from a user, wherein the step of attaching is performed based on the information in the script language, the control information, and the information from the user.

73. A method according to claim 72, wherein the step of inputting information from the user comprises graphically pointing and selecting an object on a display.

74. A method according to claim 69, further comprising the steps of inserting an object into the at least one of the data stream and file.

75. A method according to claim 74, wherein said inserting step comprises inserting an advertisement into the at least one of the data stream and file.

76. A method according to claim 75, further comprising the step of replacing the advertisement with a different object.

77. A method according to claim 74, wherein said inserting step comprises inserting a graphical character into the at least one of the data stream and file.

78. A method according to claim 77, wherein said step of inserting a graphical character comprises inserting the graphical character based on a geographical location of a user.

79. A method according to claim 69, further comprising the step of replacing one of the plurality of objects with another object.

80. A method according to claim 79, wherein said step of replacing one of the plurality of objects comprises replacing the one of the plurality of objects which is a viewed scene with a new scene.

81. A method according to claim 69, wherein said step of reading a plurality of data sources comprises reading a least one of the plurality of data sources which is training video.

82. A method according to claim 69, wherein said step of reading a plurality of data sources comprises reading a least one of the plurality of data sources which is an educational video.

83. A method according to claim 69, wherein said step of reading a plurality of data sources comprises reading a least one of the plurality of data sources which is a promotional video.

84. A method according to claim 69, wherein said step of reading a plurality of data sources comprises reading a least one of the plurality of data sources which is an entertainment video.

85. A method according to claim 69, wherein said step of reading a plurality of data sources comprises obtaining video from a surveillance camera.

86. A method according to claim 74, wherein said inserting step comprises inserting a video from a camera for viewing automobile traffic into the at least one of the data stream and file.

87. A method according to claim 74, wherein said inserting step comprises inserting information of a greeting card into the at least one of the data stream and file.

88. A method according to claim 74, wherein said inserting step comprises inserting a computer generated image of a monitor of a remote computing device.

89. A method according to claim 69, further comprising the step of providing the at least one of a data stream and a file to a user, wherein the at least one of a data stream and a file include an interactive video brochure.

90. A method according to claim 69, further comprising the step of providing the at least one of a data stream and a file which includes an interactive form to a user;
electronically filling out the form by the user; and
electronically storing information entered by the user when filling out the form.

91. A method according to claim 90, further comprising the step of transmitting the information which has been electronically stored. 92. A method according to claim 69, wherein the step of attaching control information comprises attaching control information which indicates interaction behaviour.

93. A method according to claim 69, wherein the step of attaching control information comprises attaching control information which includes rendering parameters.

94. A method according to claim 69, wherein the step of attaching control information comprises attaching control information which includes composition information.

95. A method according to claim 69, wherein the step of attaching control information comprises attaching control information which indicates how to process compressed data.

96. A method according to claim 69, wherein the step of attaching control information comprises attaching an executable behaviour.

97. A method according to claim 96, wherein the step of attaching an executable behaviour comprises attaching rendering parameters used for animation.

98. A method according to claim 96, wherein the step of attaching an executable behaviour comprises attaching a hyperlink.

99. A method according to claim 96, wherein the step of attaching an executable behaviour comprises attaching a timer.

100. A method according to claim 96, wherein the step of attaching an executable behaviour comprises attaching a behaviour which allows making a voice call.

101. A method according to claim 96, wherein the step of attaching an executable behaviour comprises attaching systems states including at least one of pause and play.

102. A method according to claim 96, wherein the step of attaching an executable behaviour comprises attaching information which allows changing of user variables.

103. A system for processing objects, comprising:
means for parsing information in a script language;
means for reading a plurality of data sources containing a plurality of objects in the form of at least one of video, graphics, animation, and audio;
means for attaching control information to the plurality of objects based on the information in the script language; and
means for interleaving the plurality of objects into at least one of a data stream and a file.

104. A system according to claim 103, further comprising means for inputting information from a user, wherein the means for attaching operates based on the information in the script language and the information from the user.

105. A system according to claim 103, further comprising means for inputting control information selected from at least one of profile information, demographic information, geographic information, and temporal information, wherein the means for attaching operates based on the information in the script language and the control information.

106. A system according to claim 105, further comprising means for inputting information from a user, wherein the means for attaching operates based on the information in the script language, the control information, and the information from the user.

107. A system according to claim 106, wherein the means for inputting information from the user comprises means for graphically pointing and selecting an object on a display.

108. A system according to claim 103, further comprising means for inserting an object into the at least one of the data stream and file.

109. A system according to claim 108, wherein said means for inserting comprises means for inserting an advertisement into the at least one of the data stream and file.

110. A system according to claim 109, further comprising means for replacing the advertisement with a different object.

111. A system according to claim 108, wherein said means for inserting comprises means for inserting a graphical character into the at least one of the data stream and file.

112. A system according to claim111, wherein said means for inserting a graphical character comprises means for inserting the graphical character based on a geographical location of a user.

113. A system according to claim 103, further comprising means for replacing one of the plurality of objects with another object.

114. A system according to claim 113, wherein said means for replacing one of the plurality of objects comprises means for replacing the one of the plurality of objects which is a viewed scene with a new scene.

115. A system according to claim 103, wherein said means for reading a plurality of data sources comprises means for reading a least one of the plurality of data sources which is a training video.

116. A system according to claim 103, wherein said means for reading a plurality of data sources comprises means for reading a least one of the plurality of data sources which is a promotional video.

117. A system according to claim 103, wherein said means for reading a plurality of data sources comprises means for reading a least one of the plurality of data sources which is an entertainment video.

118. A system according to claim 103, wherein means for reading a plurality of data sources comprises means for reading a least one of the plurality of data sources which is an educational video.

119. A system according to claim 103, wherein said means for reading a plurality of data sources comprises means for obtaining video from a surveillance camera.

120. A system according to claim 107, wherein said means for inserting comprises means for inserting a video from a camera for viewing automobile traffic into the at least one of the data stream and file.

121. A system according to claim 107, wherein said means for inserting comprises means for inserting information of a greeting card into the at least one of the data stream and file.

122. A system according to claim 107, wherein said means for inserting comprises inserting a computer generated image of a monitor of a remote computing device.

123. A system according to claim 103, further comprising means for providing the at least one of a data stream and a file to a user, wherein the at least one of a data stream and a file includes an interactive video brochure.

124. A system according to claim 103, further comprising means for providing the at least one of a data stream and a file which includes an interactive form to a user;
means for electronically filling out the form by the user; and
means for electronically storing information entered by the user when filling out the form.

125. A system according to claim 124, further comprising means for transmitting the information which has been electronically stored.

126. A system according to claim 103, wherein the means for attaching control information comprises means for attaching control information which indicates interaction behaviour.

127. A system according to claim 103, wherein the means for attaching control information comprises means for attaching control information which includes rendering parameters.

128. A system according to claim 103, wherein the means for attaching control information comprises means for attaching control information which includes composition information.

129. A system according to claim 103, wherein the means for attaching control information comprises means for attaching control information which indicates how to process compressed data.

130. A system according to claim 103, wherein the means for attaching control information comprises means for attaching an executable behaviour.

131. A system according to claim 130, wherein the means for attaching an executable behaviour comprises means for attaching rendering parameters used for animation.

132. A system according to claim 130, wherein the means for attaching an executable behaviour comprises means for attaching a hyperlink.

133. A system according to claim 130, wherein the means for attaching an executable behaviour comprises means for attaching a timer.

134. A system according to claim 130, wherein the means for attaching an executable behaviour comprises means for attaching a behaviour which allows making a voice call.

135. A system according to claim 130, wherein the means for attaching an executable behaviour comprises means for attaching systems states including at least one of pause and play.

136. A system according to claim 130, wherein the means for attaching an executable behaviour comprises means for attaching information which allows changing of user variables.

137. A method of remotely controlling a computer, comprising the stepof :
performing a computing operation at a server based on data;
generating image information at the server based on the computing operation;
transmitting, via a wireless connection, the image information from the server to a client computing device without transmitting said data;
receiving the image information by the client computing device; and
displaying the image information by the client computing device.

138. A method according to claim 137, further comprising the steps of entering, by a user of the client computing device, input information;
transmitting, via the wireless connection, the input information from the client computing device to the server;
processing the input information at the server;
altering the image information at the server based on the input information;
transmitting, via the wireless connection, the image information which has been altered;
receiving the image information which has been altered by the client computing device; and
displaying the image information which has been altered by the client computing device.

139. A method according to claim 137, further comprising the step of capturing the image information at the server, wherein the transmitting step comprises transmitting the image information which has been captured.

140. A method according to claim 137, wherein the transmitting step comprises transmitting the image information as a video object having attached thereto control information.

141. A system for remotely controlling a computer, comprising:
means for performing a computing operation at a server based on data;
means for generating image information at the server based on the computing operation;
means for transmitting, via a wireless connection, the image information from the server to a client computing device without transmitting said data;
means for receiving the image information by the client computing device; and
means for displaying the image information by the client computing device.

142. A system according to claim 141, further comprising means for entering, by a user of the client computing device, input information;
means for transmitting, via the wireless connection, the input information from the client computing device to the server;
means for processing the input information at the server;
means for altering the image information at the server based on the input information;
means for transmitting, via the wireless connection, the image information which has been altered;
means for receiving the image information which has been altered by the client computing device; and
means for displaying the image information which has been altered by the client computing device.

143. A system according to claim 141, further comprising means for capturing the image information at the server,
wherein the means for transmitting comprises:
means for transmitting the image information which has been captured.

144. A system according to claim 139, wherein the means for transmitting comprises means for transmitting the image information as a video object having attached thereto control information.

145. A method of transmitting an electronic greeting card, comprising the stepsof :
inputting information indicating features of a greeting card;
generating image information corresponding to the greeting card;
encoding the image information as an object having control information;
transmitting the object having the control information over a wireless connection;
receiving the object having the control information by a wireless hand-held computing device;
decoding the object having the control information into a greeting card image by the wireless hand-held computing device; and
displaying the greeting card image which has been decoded on the hand-held computing device.

146. A method according to claim 145, wherein the step of generating image information comprises capturing at least one of an image and as series of images as custom image information, wherein the encoding step further comprises encoding said custom image as an object having control information, wherein said step of decoding comprises decoding the object encoded using the image information and decoding the object encoded using the custom image information, wherein said displaying step comprises displaying image information and the custom image information as the greeting card.

147. A system transmitting an electronic greeting card, comprising:
means for inputting information indicating features of a greeting card;
means for generating image information corresponding to the greeting card;
means for encoding the image information as an object having control information;
means for transmitting the object having the control information over a wireless connection;
means for receiving the object having the control information by a wireless handheld computing device;
means for decoding the object having the control information into a greeting card image by the wireless hand-held computing device; and
means for displaying the greeting card image which has been decoded on the handheld computing device.

148. A system according to claim 147, wherein the means for generating image information comprises means for capturing at least one of an image and as series of images as custom image information, wherein the means for encoding further comprises means for encoding said custom image as an object having control information, wherein said means for decoding comprises means for decoding the object encoded using the image information and decoding the object encoded using the custom image information, wherein said means for displaying comprises means for displaying image information and the custom image information as the greeting card.

149. A method of controlling a computing device, comprising the stepsof :
inputting an audio signal by a computing device;
encoding the audio signal;
transmitting the audio signal to a remote computing device;
interpreting the audio signal at the remote computing device and generating information corresponding to the audio signal;
transmitting the information corresponding to the audio signal to the computing device;
controlling the computing device using the information corresponding to the audio signal.

150. A method according to claim 149, wherein said controlling step comprises controlling the computing device using computer instructions which corresponds to the information corresponding to the audio signal.

151. A method according to claim 149, wherein said controlling step comprises controlling the computing device using data which corresponds to the information corresponding to the audio signal.

152. A method according to claim 149, wherein the step of interpreting the audio signal comprises performing a speech recognition.

153. A system for controlling a computing device, comprising:
inputting an audio signal by a computing device;
encoding the audio signal;
transmitting the audio signal to a remote computing device;
interpreting the audio signal at the remote computing device and generating information corresponding to the audio signal;
transmitting the information corresponding to the audio signal to the computing device; and
controlling the computing device using the information corresponding to the audio signal.

154. A system according to claim 153, wherein said means for controlling comprises means for controlling the computing device using computer instructions which corresponds to the information corresponding to the audio signal.

155. A system according to claim 153, wherein said means for controlling comprises means for controlling the computing device using data which corresponds to the information corresponding to the audio signal.

156. A system according to claim 153, wherein said means for interpreting the audio signal comprises means for performing a speech recognition.

157. A method of performing a transmission, comprising the stepsof :
displaying an advertisement on a wireless hand-held device;
transmitting information from the wireless hand-held device; and

receiving a discounted price associated with the information which has been transmitted because of the display of the advertisement.

158. A method according to claim 157, wherein the displaying step is performed before the transmitting step.

159. A method according to claim 157, wherein the displaying step is performed during the transmitting step.

160. A method according to claim 157, wherein the displaying step is performed after the transmitting step.

161. A method according to claim 157, wherein the step of receiving a discounted price comprises receiving a discount of an entire cost associated with the information which has been transmitted.

162. A method according to claim 157, wherein the step of displaying comprises displaying the object as an interactive object, the method further comprising interacting with the object by the user; and displaying a video in response to interacting by the user.

163. A system for performing a transmission, comprising:
means for displaying an advertisement on a wireless hand-held device ;
means for transmitting information from the wireless hand-held device; and
means for receiving a discounted price associated with the information which has been transmitted because of the display of the advertisement.

164. A system according to claim 163, wherein the means for displaying the advertisement operates before the transmitting of information.

165. A system according to claim 163, wherein the means for displaying the advertisement operates during the transmitting of ~ 166. A system according to claim 163, wherein the means for displaying the advertisement operates after the transmitting of information.

167. A system according to claim 163, wherein the means for receiving a discounted price comprises means for receiving a discount of an entire cost associated with the information which has been transmitted.

168. A system according to claim 163, wherein the means for displaying comprises means for displaying the object as an interactive object, the system further comprising means for interacting with the object by the user; and means for displaying a video in response to interacting by the user.

169. A method of providing video, comprising the stepsof :
determining whether an event has occurred; and
obtaining a video of an area transmitting to a user by a wireless transmission the video of the area in response to the event.

170. A method according to claim 169, wherein the step of determining comprises selecting a location by the user, wherein the step of transmitting comprises transmitting the video of the area which corresponds to said location.

171. A method according to claim 170, wherein the step of selecting comprises dialing a phone number corresponding to traffic video.

172. A method according to claim 169, further comprising the step of performing a determination of the area using a global position system.

173. A method according to claim 169, further comprising the step of performing a determination of the area based on a cell site utilized by the user.

174. A method according to claim 169, wherein the step of determining comprises determining that a traffic problem exists on a predefined route, wherein the step of obtaining video comprises obtaining video which corresponds to the predefined route.

175. A method according to claim 174, wherein the step of transmitting comprises transmitting the video to the user only when the user is moving greater than a predetermined speed.

176. A system for providing video, comprising:
means for determining whether an event has occurred;
means for obtaining a video of an area; and
means for transmitting to a user by a wireless transmission the video of the area in response to the event.

177. A system according to claim 176, wherein the means for determining comprises means for selecting a location by the user, wherein the means for transmitting comprises means for transmitting the video of the area which corresponds to said location.

178. A system according to claim 177, wherein the means for selecting comprises means for dialing a phone number corresponding to traffic video.

179. A system according to claim 176, further comprising means for performing a determination of the area using a global position system.

180. A system according to claim 176, further comprising means for performing a determination of the area based on a cell site utilized by the user.

181. A system according to claim 176, wherein the means for determining comprises means for determining that a traffic problem exists on a predefined route, wherein the means for obtaining video comprises means for obtaining video which corresponds to the predefined route.

182. A system according to claim 181, wherein the means for transmitting comprises means for transmitting the video to the user only when the user is moving greater than a predetermined speed.

183. An object oriented multimedia video system capable of supporting multiple arbitrary shaped video objects without the need for extra data overhead or processing overhead to provide video object shape information.

184. A system according to claim 183, wherein said video objects have their own attached control information.

185. A system according to claim 183, wherein said video objects are streamed from a remote server to a client.

186. A system according to claim 183, wherein said video object shape is intrinsically encoded in the representation of the images.

187. A method according to claim 69, wherein the step of attaching control information comprises attaching conditions for execution of controls.

188. A method according to claim 71 further comprising the steps of obtaining information from user flags or variables, wherein the step of attaching is performed based on the information in the script language, the control information, and the information from said user

flags.

189. A method of delivering multimedia content to wireless devices by server initiated communications wherein content is scheduled for delivery at a desired time or cost effective manner and said user is alerted to completion of delivery via device's display or other indicator.

190. A method according to claim 189, wherein said user registers a request for delivery of specific content with a content service provider, said request being used to automatically schedule network initiated delivery to the client device.

191. An interactive system wherein stored information can be viewed offline and stores user input and interaction to be automatically forwarded over a wireless network to a specified remote server when said device next connects online.

192. An interactive system according to claim 191, wherein said stored information is object oriented multimedia data which can be navigated non-linearly.

193. A method according to claim 69, wherein said step of reading a plurality of data sources comprises reading a least one of the plurality of data sources which take the form of marketing, promotional, product information, entertainment videos videos.

194. A method according to claim51, wherein the encoding step comprises creating the quadtree to have leaf node values represented as an index into a FIFO buffer if a flag is defined true or as the colour value if said the flag is false.

195. A system according to claim 66, wherein the means for encoding comprises means for creating the quadtree to have leaf node values represented as an index into a FIFO buffer if a flag is defined true or as the colour value if said the flag is false.

196. A method according to claim51, wherein the encoding step comprises creating the quadtree to have leaf node values represented as the mean plus horizontal and vertical gradients.

197. A method according to claim 196, wherein the encoding step comprises creating the quadtree to have leaf node mean values represented as an index into a FIFO buffer if a flag is defined true or as the colour value if said the flag is false.

198. A system according to claim 66, wherein the means for encoding comprises creating the quadtree to have leaf node values represented as the mean plus horizontal and vertical gradients.

199. A system according to claim 198, wherein the means for encoding comprises creating the quadtree to have leaf node mean values represented as an index into a FIFO buffer if a flag is defined true or as the colour value if said the flag is false.

200. A system according to claim 14 including a persistent object library on a portable client device for use in dynamic media composition said library being capable of being managed from said remote server, software available to a client for executing library management instructions delivered to it from said remote server, said server capable of querying said library and receiving information about specific objects contained therein, and inserting, updating, or deleting the contents of said library; and said dynamic media composition engine capable of sourcing object data stream simultaneously both from said library and remote server, if required, said persistent object library storing object information including expiry dates, access permissions, unique identifiers, metadata and state information, said system performing automatic garbage collection on expired objects, access control, library searching, and various other library management tasks.

201. A video encoding method, including:
encoding video data with object control data as a video object; and
generating a data stream including a plurality of said video object with respective video data and object control data.

202. A video encoding method as claimed in claim 201, including:
generating a scene packet representative of a scene and including a plurality of said data stream with respective video objects.

203. A video encoding method as claimed in claim 202, including generating a video data file including a plurality of said scene packet with respective data streams and user control data.

204. A video encoding method as claimed in claim 201, wherein said video data represents video frames, audio frames, text and/or graphics.

205. A video encoding method as claimed in claim 201, wherein said video object comprises a packet with data packets of said encoded video data and at least one object control packet with said object control data for said video object.

206. A video encoding method as claimed in claim 202, wherein said video data file, said scene packets and said data streams include respective directory data.

207. A video encoding method as claimed in claim 201, wherein said object control data represents parameters defining said video object to allow interactive control of said object within a scene by a user.

208. A video encoding method as claimed in claim 201, wherein said encoding includes encoding luminance and colour information of said video data with shape data representing the shape of said video object.

209. A video encoding method as claimed in claim201, wherein said object control data defines shape, rendering, animation and interaction parameters for said video objects.

210. A video encoding method, including:
quantising colour data in a video stream based on a reduced representation of colours;
generating encoded video frame data representing said quantised colours and transparent regions; and
generating encoded audio data and object control data for transmission with said encoded video data.

211. A video encoding method as claimed in claim 210, including:
generating motion vectors representing colour changes in a video frame of said stream; said encoded video frame data representing said motion vectors.

212. A video encoding method as claimed in claim 211, including:
generating encoded text object and

215. A video encoding method as claimed in claim 210 or 211, wherein said object control data represents parameters for rendering objects of a video frame.

216. A video encoding method as claimed in claim 210 or211, wherein said parameters represents transparency, scale, volume, position, and rotation.

217. A video encoding method as claimed in claim210 or 211, wherein said encoded video, audio and control data are transmitted as

respective packets for respective decoding.

218. A video encoding method, including:
(i) selecting a reduced set of colours for each video frame of video data;
(ii) reconciling colours from frame to frame ;
(iii) executing motion compensation;
(iv) determining update areas of a frame based on a perceptual colour difference measure;
(v) encoding video data for said frames into video objects based on steps (i) to (iv); and
(vi) including in each video object animation, rendering and dynamic composition controls.

219. A video decoding method for decoding video data encoded according to a method as claimed in any one of the preceding claims.

220. A video decoding method as claimed in claim 219, including parsing said encoded data to distribute object control packets to an object management process and encoded video packets to a video decoder.

221. A video encoding method as claimed in claim 214, wherein said rendering parameters represent object transparency, scale, volume, position and rotation.

*222. A video encoding method as claimed in claim 214, wherein said animation paths adjust said rendering parameters.

223. A video encoding method as claimed in claim 214, wherein said hyperlinks represent links to respective video files, scene packets and objects.

224. A video encoding method as claimed in claim 214, wherein said interactive behaviour data provides controls for play of said objects, and return of user data.

225. A video decoding method as claimed in claim 220 including generating video object controls for a user based on said object control packets for received and rendered video objects.

226. A video decoder having components for executing the steps of the video decoding method as claimed in claim 219.

227. A computer device having a video decoder as claimed in claim 226.

228. A computer device as claimed in claim 227, wherein said device is portable and handheld, such as a mobile phone or PDA.

229. A dynamic colour space encoding method including executing the video encoding method as claimed in claim 1 and adding additional colour quantisation information for transmission to a user to enable said user to select a real-time colour reduction.

230. A video encoding method as claimed in claim 201, including adding targeted user and/or local video advertising with said video object.

231. A computer device having an ultrathin client for executing the video decoding method as claimed in claim 219 and adapted to access a remote server including said video objects.

232. A method of multivideo conferencing including executing the video encoding method as claimed in claim 201.

233. A video encoding method as claimed in claim 201, including generating video menus and forms for user selections for inclusion in said video objects.

234. A method of generating electronic cards for transmission to mobile phones including executing said video encoding method as claimed in claim201.

235. A video encoder having components for executing the steps of the video encoding method as claimed in any one of claims 201 to 218.

236. A video on demand system including a video encoder as claimed in claim 235.

237. A video security system including a video encoder as claimed in claim 235.

238. An interactive mobile video system including a video decoder as claimed in claim 226.

239. A video decoding method as claimed in 219 including processing voice commands from a user to control a video display generated on the basis of said video objects.

240. A computer program stored on a computer readable storage medium including code for executing a video decoding method as claimed in claim 219 and generating a video display including controls for said video objects, and adjusting said display in response to application of said controls.

241. A computer program as claimed in claim 240 including IAVML instructions.

242. A wireless streaming video and animation system, including:
(i) a portable monitor device and first wireless communication means;
(ii) a server for storing compressed digital video and computer animations and enabling a user to browse and select digital video to view from a library of available videos; and
(iii) at least one interface module incorporating a second wireless communication means for transmission of transmittable data from the server to the portable monitor device, the portable monitor device including means for receiving said transmittable data, converting the transmittable data to video images displaying the video images, and permitting the user to communicate with the server to interactively browse and select a video to view.

243. A wireless streaming video and animation system as claimed in claim 242, wherein said portable wireless device is a hand held processing device.

244. A method of providing wireless streaming of video and animation including at least one of the stepsof :
(a) downloading and storing compressed video and animation data from a

remote server over a wide area network for later transmission from a local server;
(b) permitting a user to browse and select digital video data to view from a library of video data stored on the local server;
(c) transmitting the data to a portable monitor device; and
(d) processing the data to display the image on the portable monitor device.

245. A method of providing an interactive video brochure including at least one of the stepsof :
(a) creating a video brochure by specifying (i) the various scenes in the brochure and the various video objects that may occur within each scene, (ii) specifying the preset and user selectable scene navigational controls and the individual composition rules for each scene, (iii) specifying rendering parameters on media objects, (iv) specifying controls on media objects to create forms to collect user feedback, (v) integrating the compressed media streams and object control information into a composite data stream.

246. A method as claimed in claim 245, including:
(a) processing the composite data stream and interpreting the object control information to display each scene;
(b) processing user input to execute any relevant object controls, such as navigation through the brochure, activating animations etc, registering and user selections and other user input;
(c) storing the user selections and user input for later uploading to the provider of the video brochures network server when a network connection becomes available; and
(d) at a remote network server, receiving uploads of user selections from interactive video brochures and processing the information to integrate it into a customer/client database.

247. A method of creating and sending video greeting cards to mobile devices including at least one of the stepsof :
(a) permitting a customer to create the video greeting card by (i) selecting a template video scene or animation form a library, (ii) customising the template by adding user supplied text or audio objects or selecting video objects from a library to be inserted as actors in the scene;
(b) obtaining from the customer (i) identification details, (ii) preferred delivery method, (iii) payment details, (iv) the intended recipient's mobile device number; and
(c) queuing the greeting card depending on the nominated delivery method until either bandwidth becomes available or off peak transport can be obtained, polling the recipient's device to see if it is capable of processing the greeting card and if so forwarding to the nominated mobile device. 248. A video encoding method as claimed in claim 201, wherein said object control data includes shape parameters that allow a user to render arbitrary shape video corresponding to said video object.

249. A video encoding method as claimed in claim 201, wherein said object control data includes condition data determining when to invoke corresponding controls for said video object.

250. A video encoding method as claimed in claim 201, wherein said object control data represents controls for affecting another video object.

251. A video encoding method as claimed in claim 201, including controlling dynamic media composition of said video objects on the basis of flags set in response to events or user interactions.

252. A video encoding method as claimed in claim 201, including broadcasting and/or multicasting said data stream.

1/48

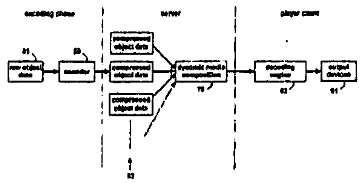


Figure 1

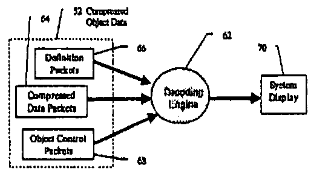


Figure 2

2/48

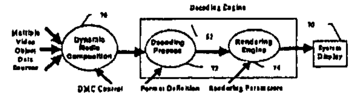


Figure 3

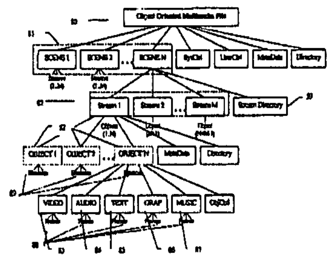


Figure 4

3/48

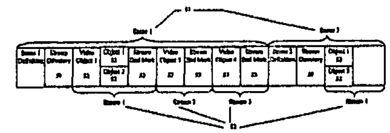


Figure 5

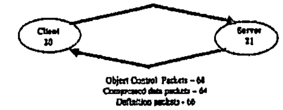


Figure 6

4/48

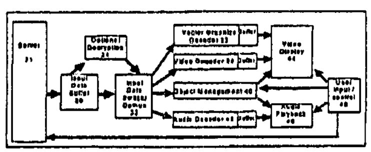


Figure 7

5/48

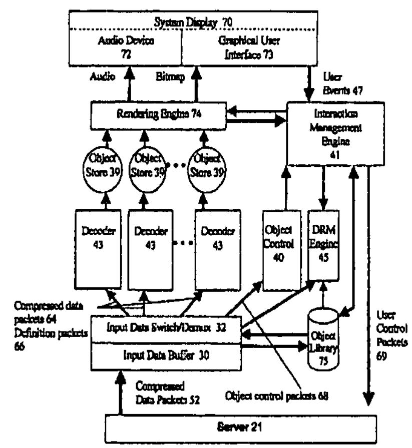


Figure 8

6/48

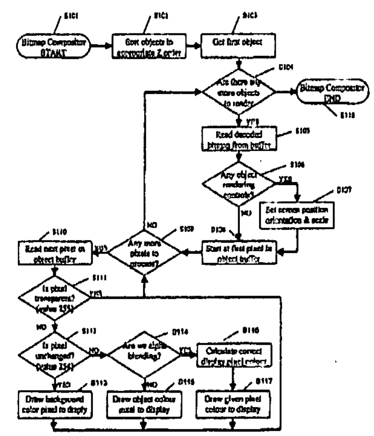


Figure 9

7/48

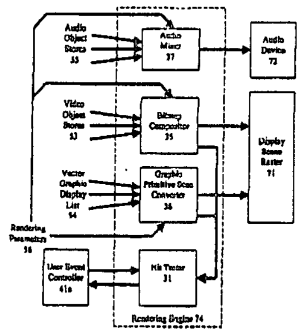


Figure 10

8/48

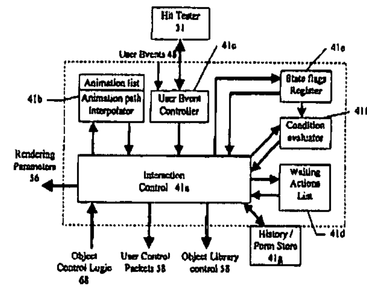


Figure 11

9/48

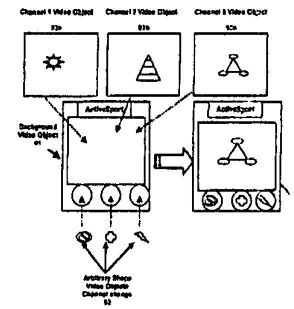


Figure 12

10/48

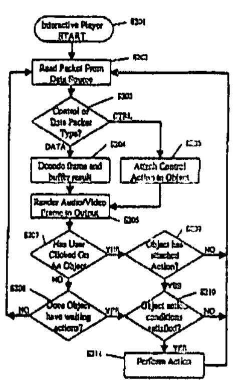


Figure 13

11/48

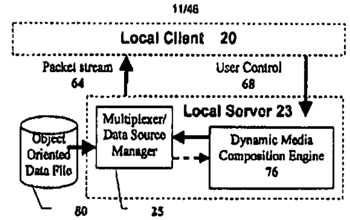


Figure 14

12/48

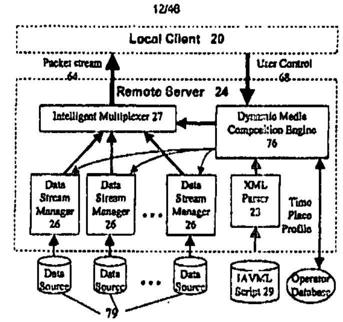
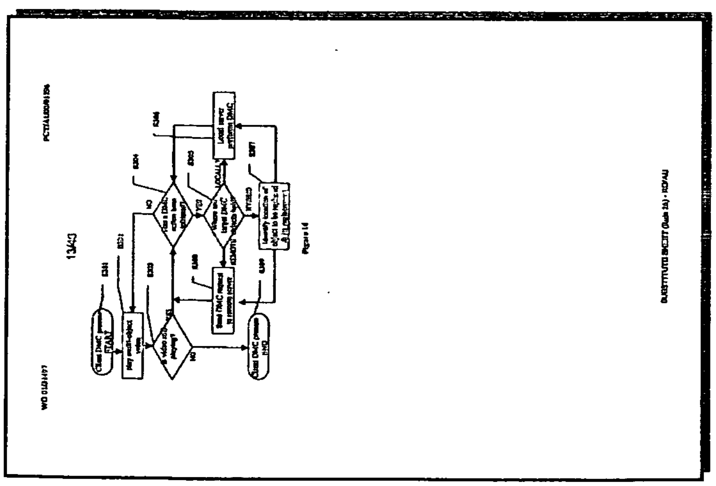
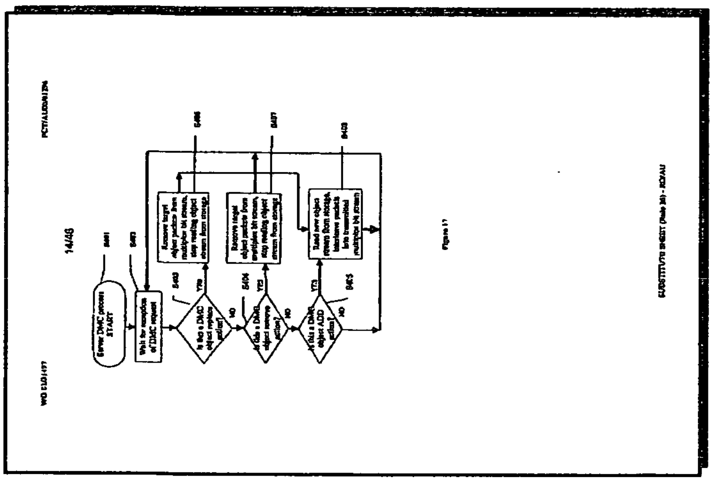
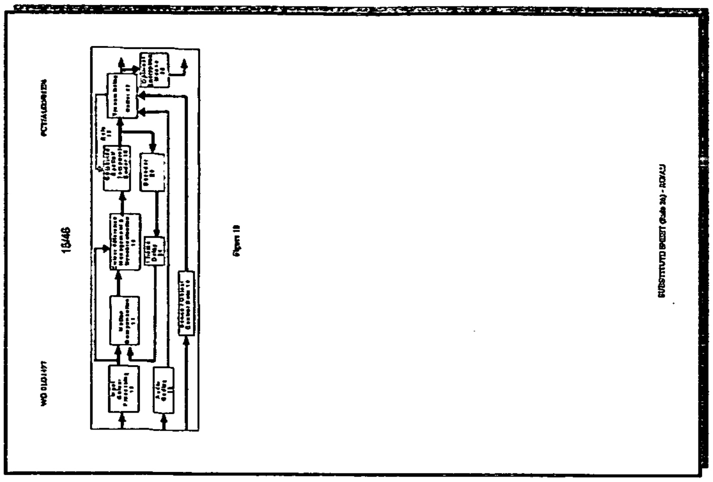
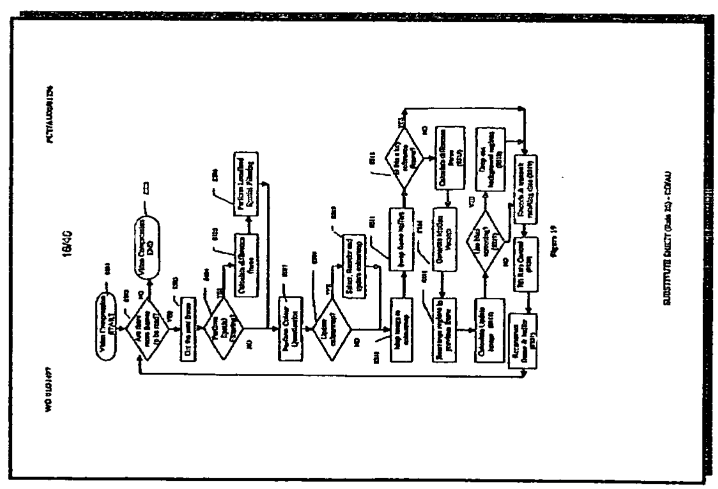


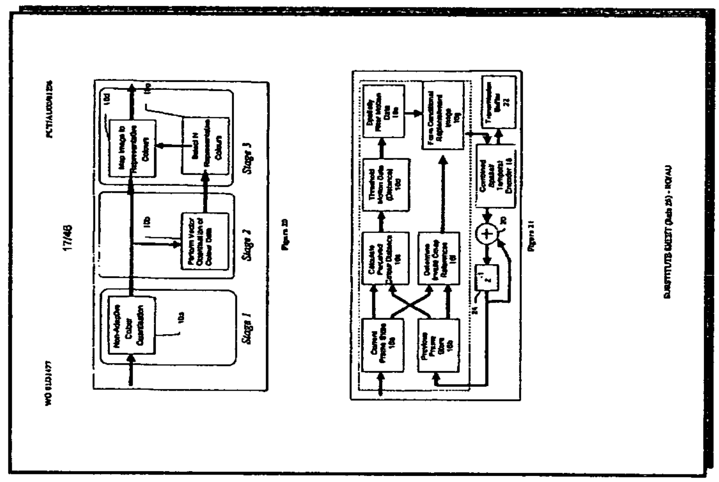
Figure 15

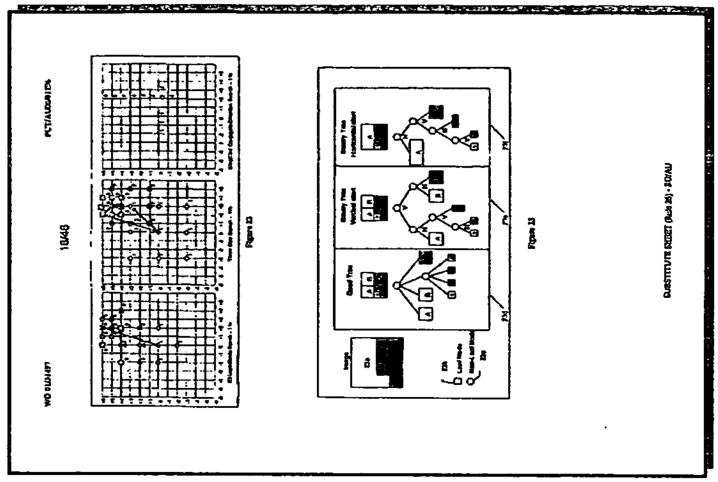












19/46

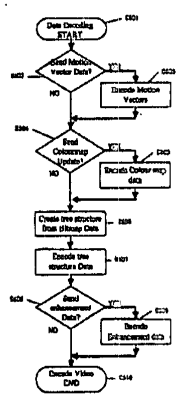


Figure 24

20/46

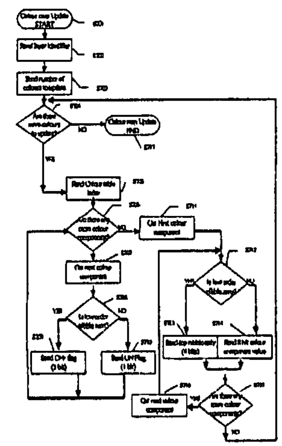


Figure 25

21/46

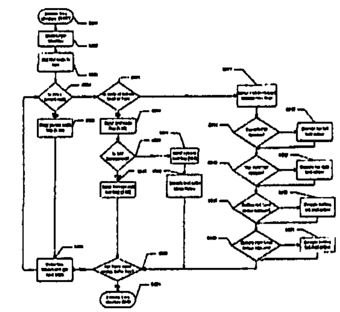


Figure 26

22/46

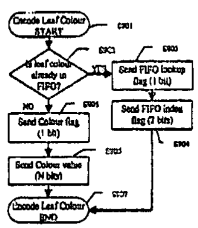


Figure 27

23/46

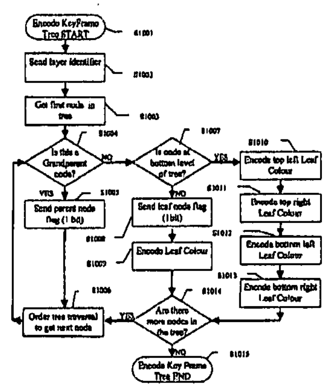


Figure 28

24/46

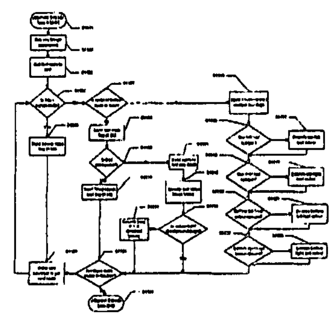
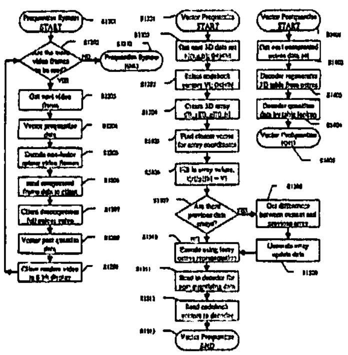


Figure 29

25/46



SUBSTITUTE SHEET (Rule 26) - RO/AU

26/46

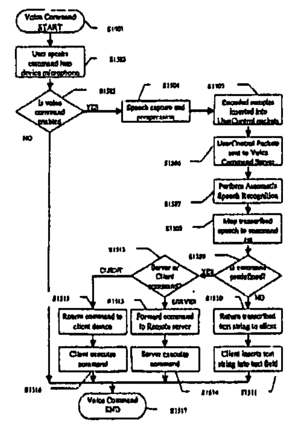


Figure 31

SUBSTITUTE SHEET (Rule 26) - RO/AU

27/46

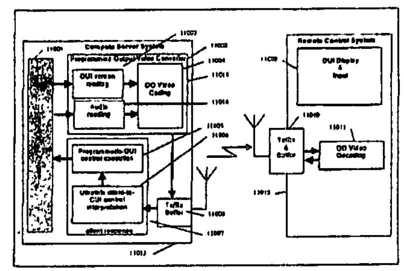


Figure 33

SUBSTITUTE SHEET (Rule 26) - RO/AU

28/46

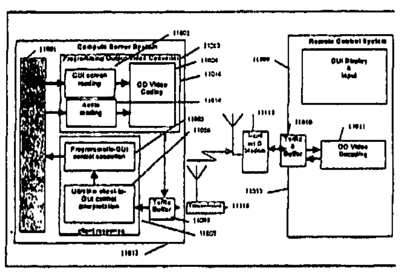


Figure 33

SUBSTITUTE SHEET (Rule 26) - RO/AU

29/46

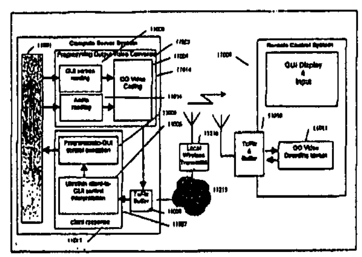


Figure 34

SUBSTITUTE SHEET (Rule 26) - RO/AU

30/46

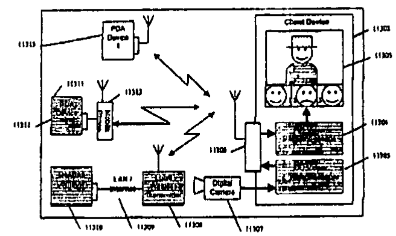


Figure 35

SUBSTITUTE SHEET (Rule 26) - RO/AU

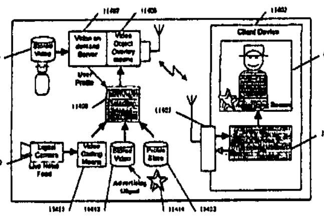


Figure 36

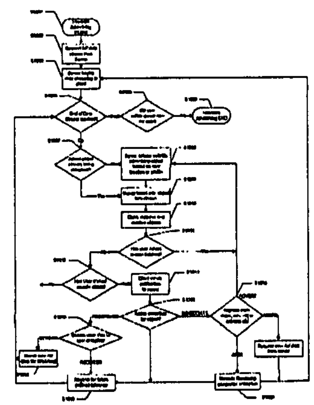


Figure 37

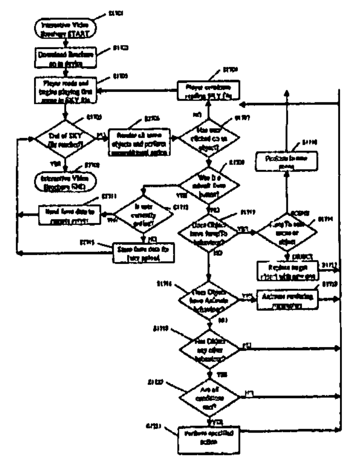


Figure 38

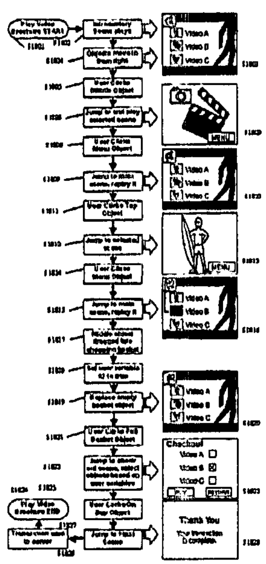


Figure 39

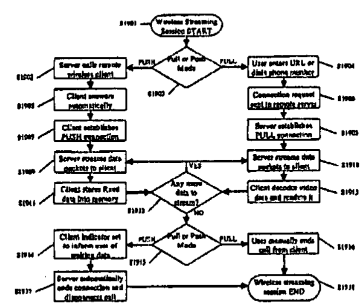


Figure 40

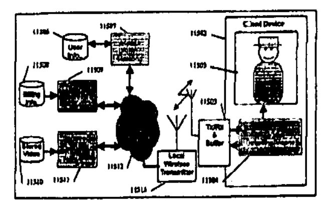
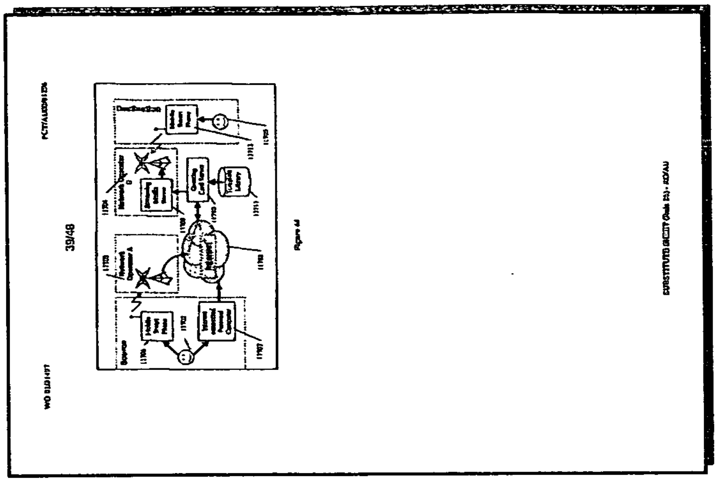
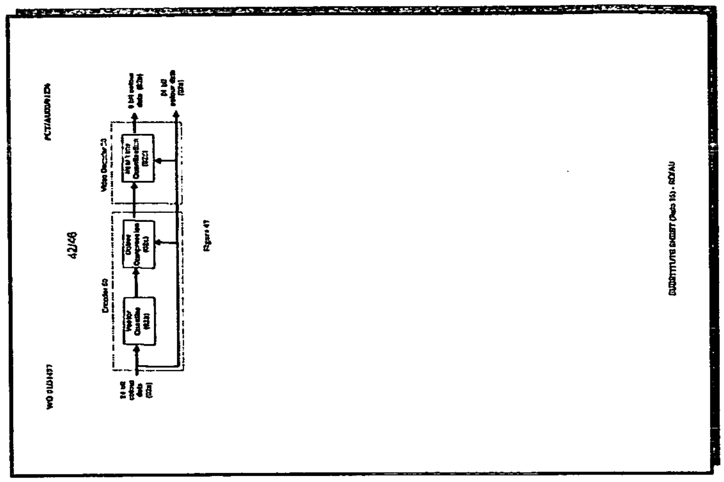
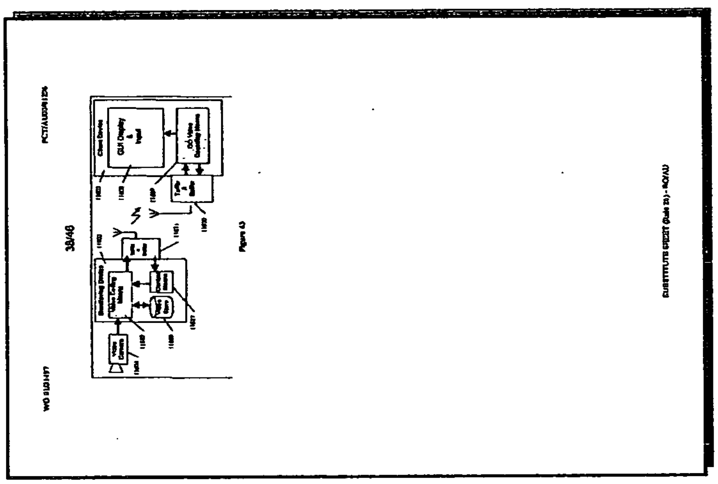
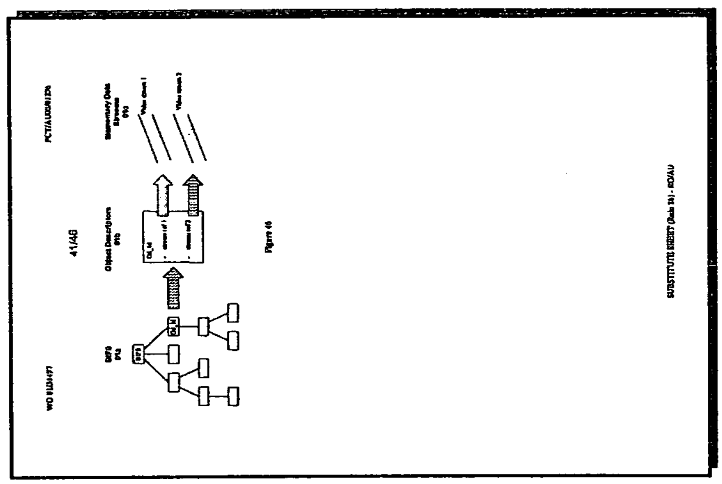


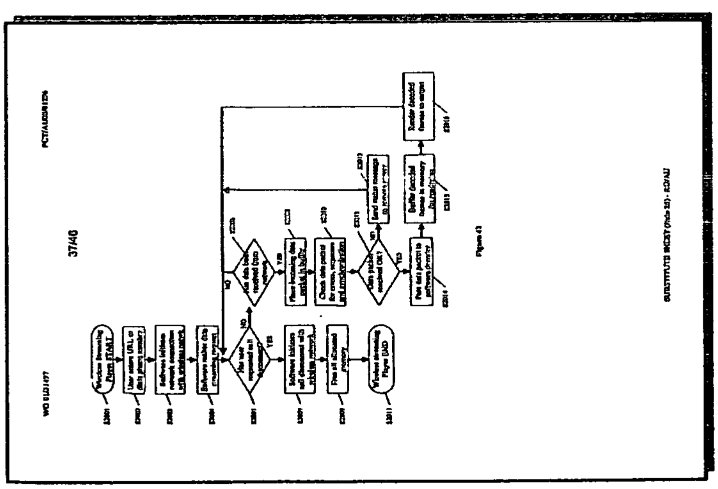
Figure 41

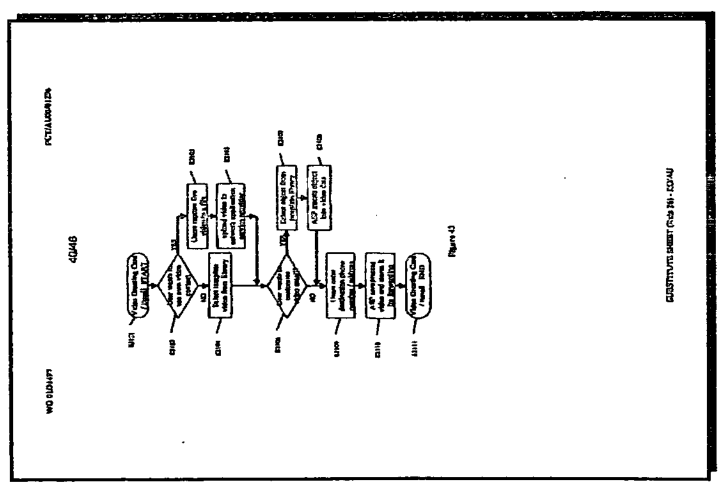

39/48


42/48


38/48


41/48


37/48


40/48

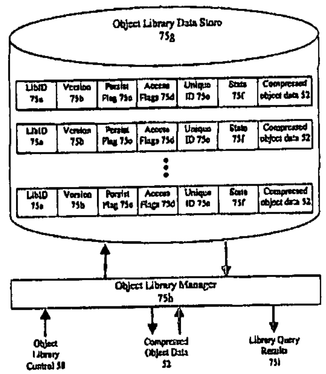


Figure 42

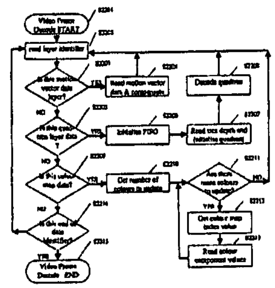


Figure 49

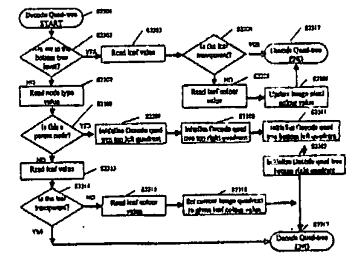


Figure 50

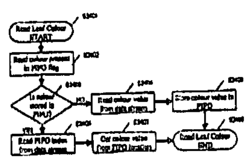


Figure 51

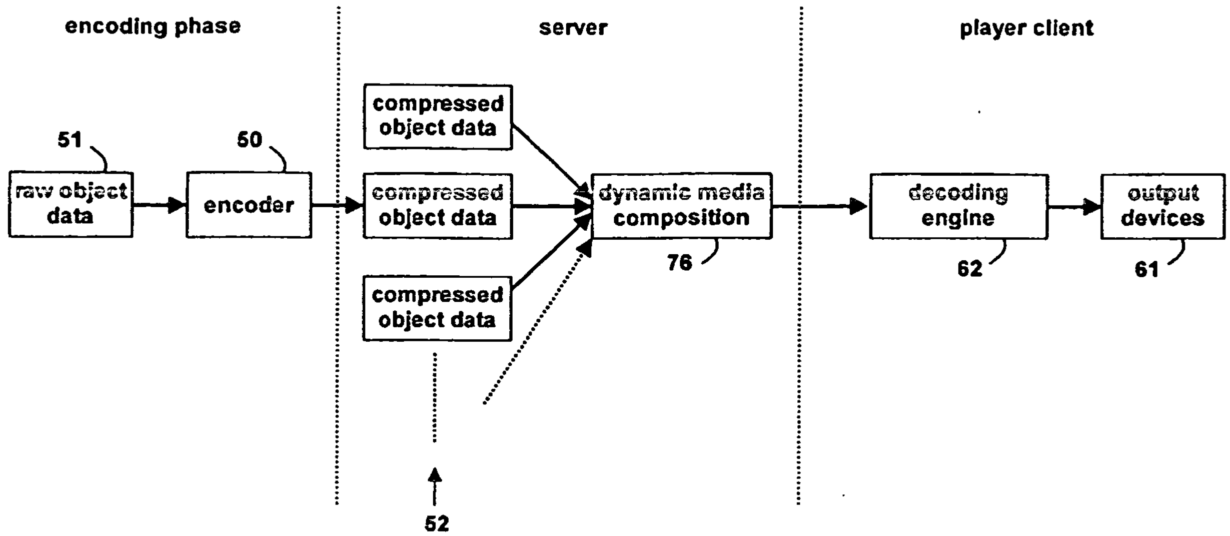


Figure 1

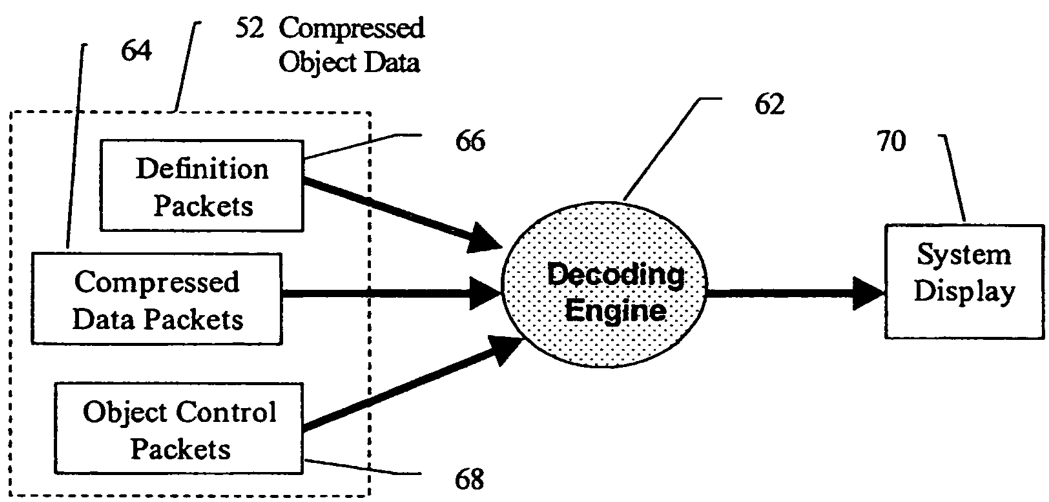


Figure 2

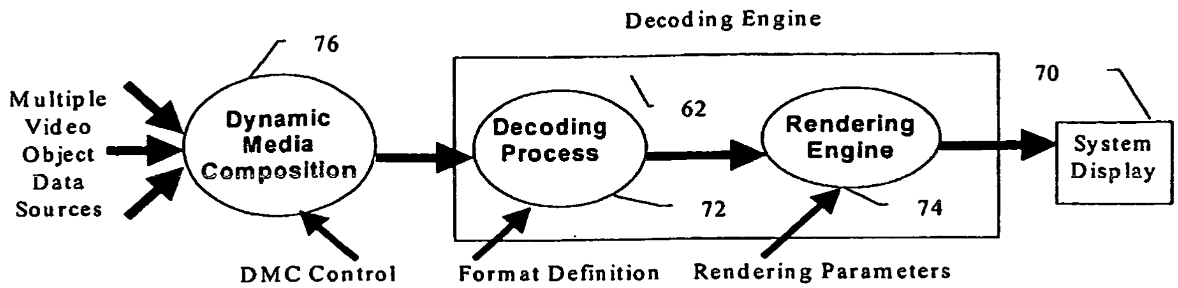


Figure 3

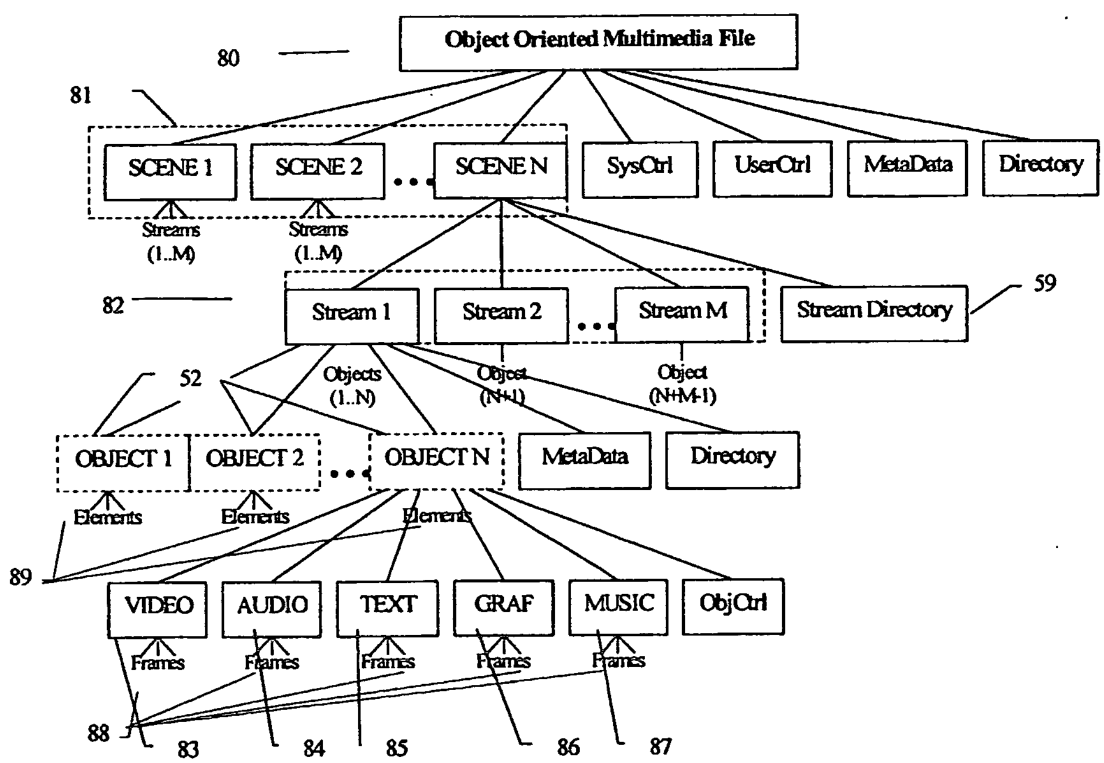


Figure 4

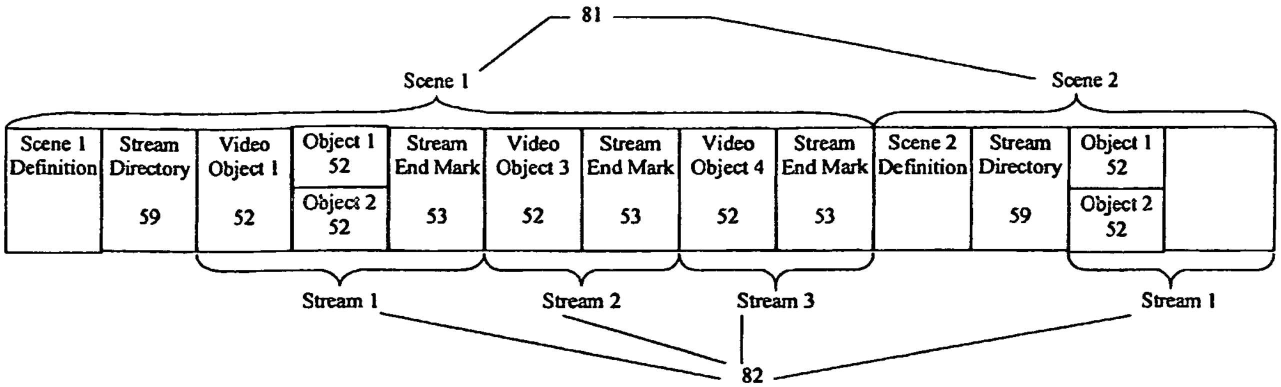


Figure 5

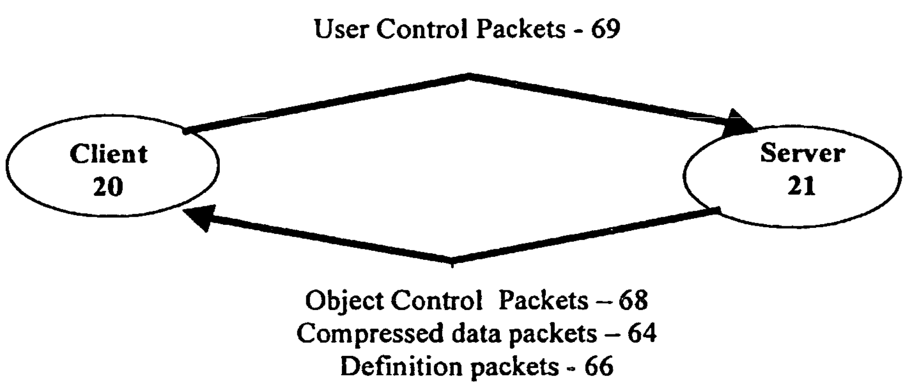


Figure 6

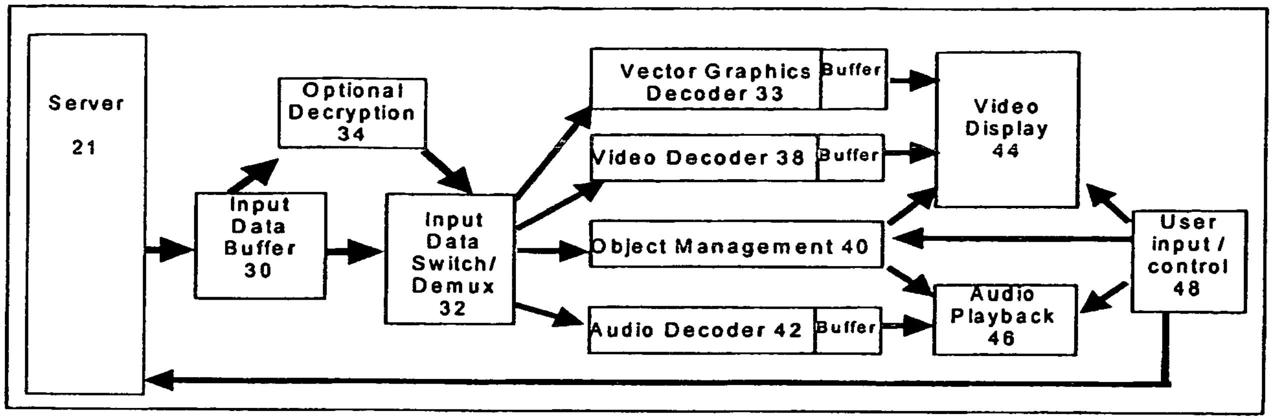

Server
21
Optional
Decryption
34
Input
Data
Buffer
30
Input
Data
Switch/
Demux
32
Vector Graphics
Decoder 33
Buffer
Video Decoder 38
Buffer
Object Management 40
Audio Decoder 42
Buffer
Video
Display
44
Audio
Playback
46
User
input /
control
48

Figure 7

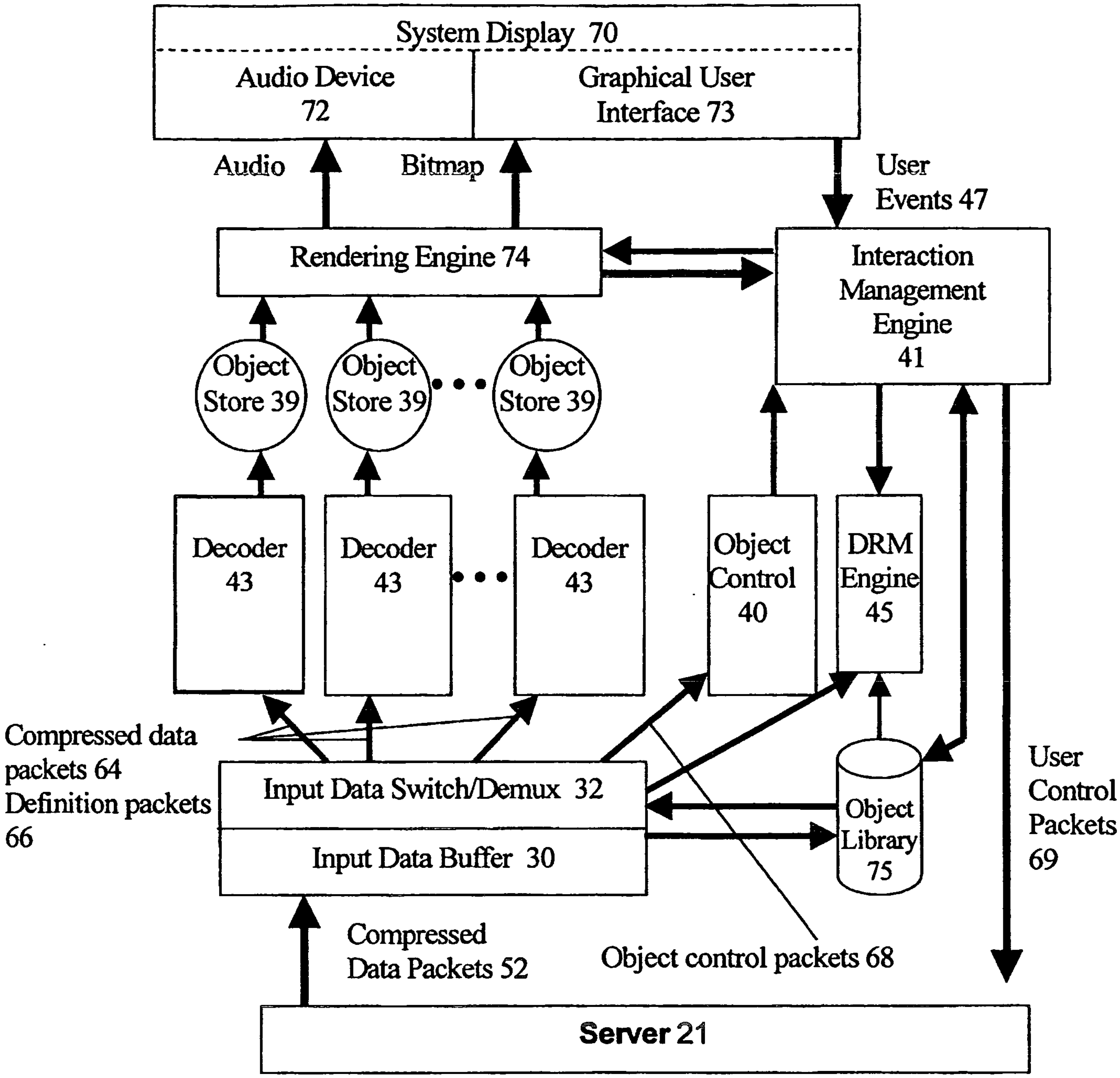

System Display 70
Audio Device 72
Graphical User Interface 73
Audio
Bitmap
User Events 47
Rendering Engine 74
Interaction Management Engine 41
Object Store 39
Object Store 39
Object Store 39
Decoder 43
Decoder 43
Decoder 43
Object Control 40
DRM Engine 45
Compressed data packets 64
Definition packets 66
Input Data Switch/Demux 32
Input Data Buffer 30
Object Library 75
User Control Packets 69
Compressed Data Packets 52
Object control packets 68
Server 21

Figure 8

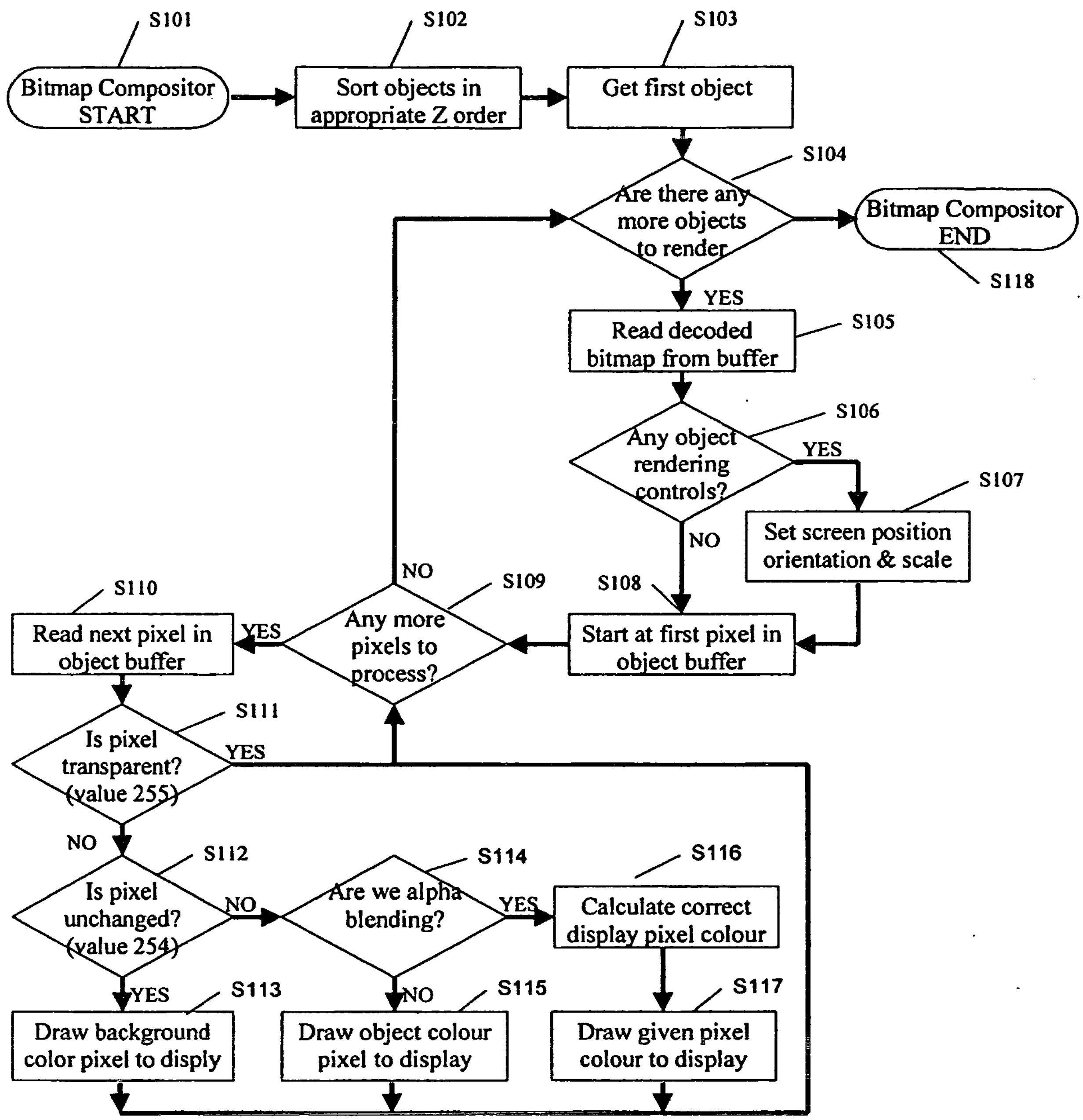

S101
Bitmap Compositor START
S102
Sort objects in appropriate Z order
S103
Get first object
S104
Are there any more objects to render
Bitmap Compositor END
S118
YES
Read decoded bitmap from buffer
S105
S106
Any object rendering controls?
YES
S107
Set screen position orientation & scale
NO
S108
Start at first pixel in object buffer
NO
S109
Any more pixels to process?
YES
S110
Read next pixel in object buffer
S111
Is pixel transparent? (value 255)
YES
NO
S112
Is pixel unchanged? (value 254)
NO
S114
Are we alpha blending?
YES
S116
Calculate correct display pixel colour
YES
S113
Draw background color pixel to disply
NO
S115
Draw object colour pixel to display
S117
Draw given pixel colour to display

Figure 9

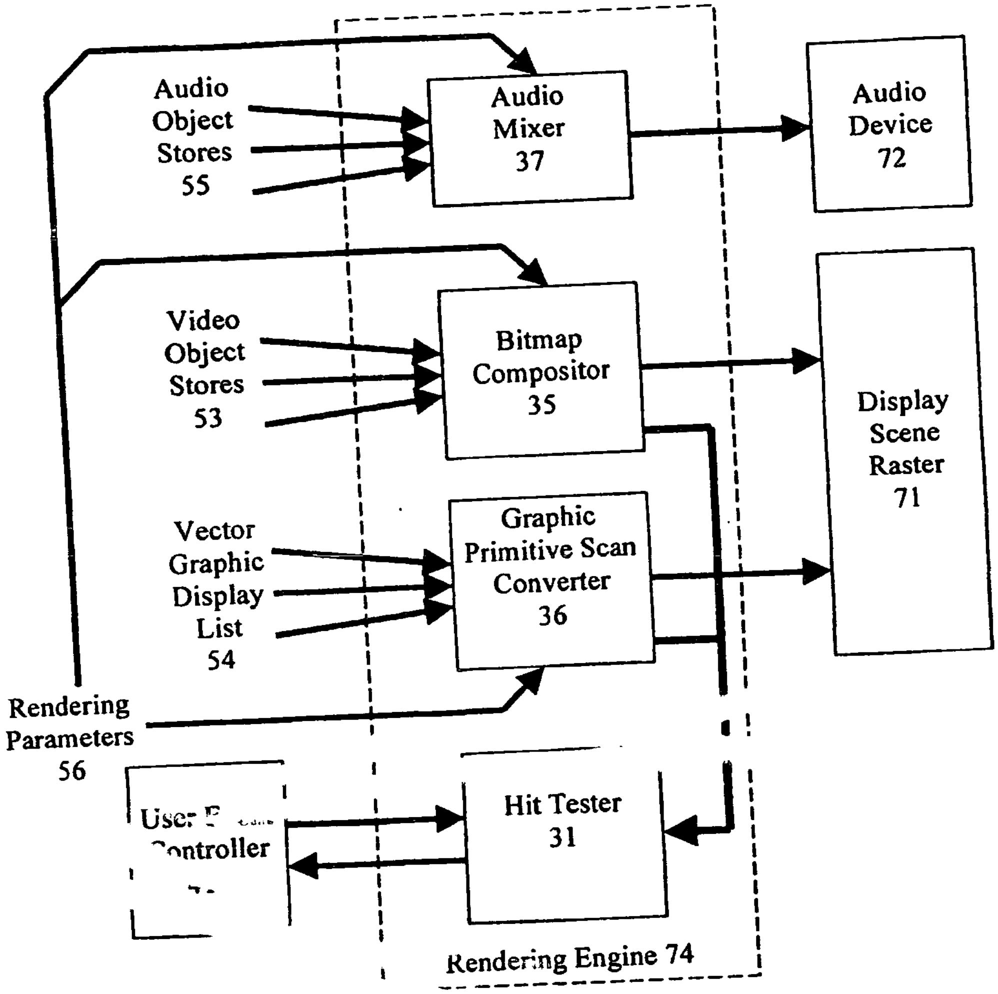

Audio Object Stores 55
Audio Mixer 37
Audio Device 72
Video Object Stores 53
Bitmap Compositor 35
Display Scene Raster 71
Vector Graphic Display List 54
Graphic Primitive Scan Converter 36
Rendering Parameters 56
Hit Tester 31
Rendering Engine 74

Figure 10

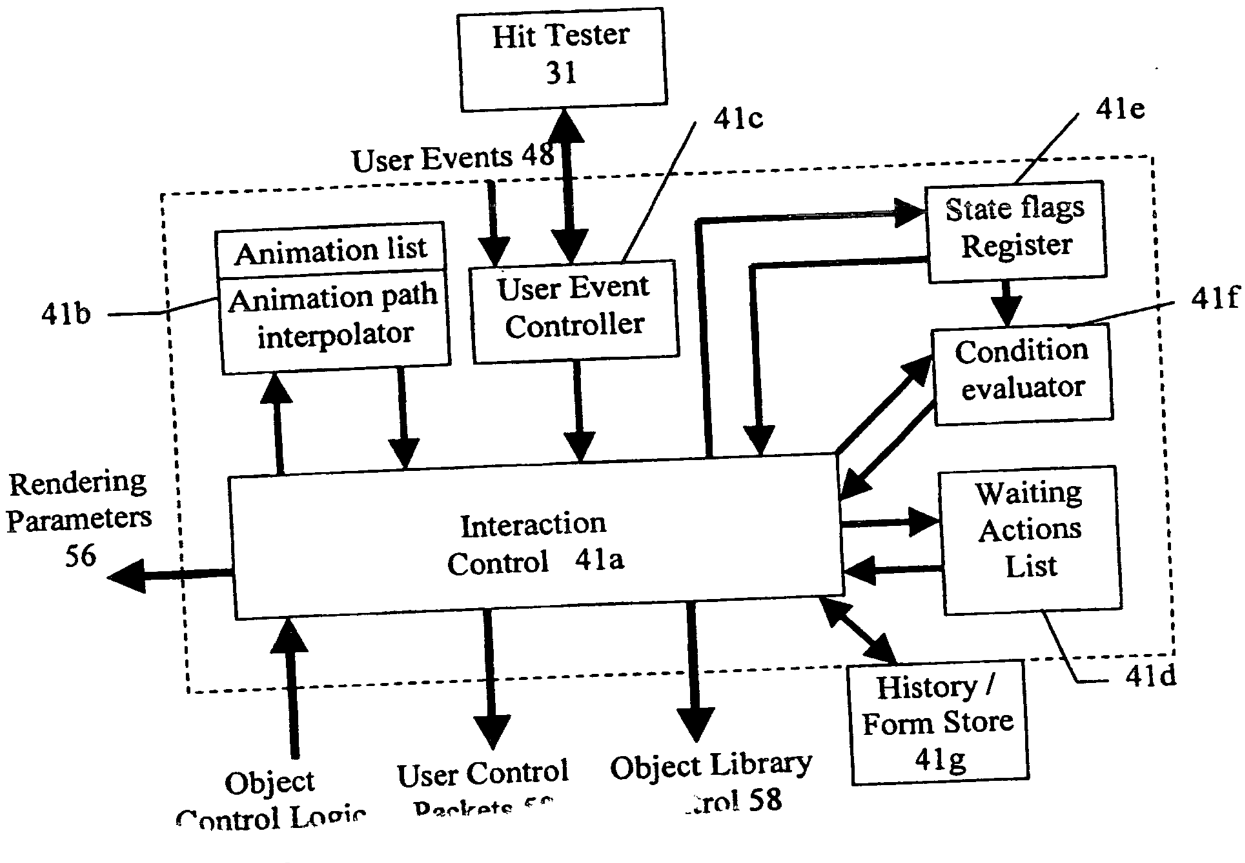

Hit Tester 31
User Events 48
41c
41e
State flags Register
Animation list
Animation path interpolator
User Event Controller
41b
41f
Condition evaluator
Rendering Parameters 56
Interaction Control 41a
Waiting Actions List
History / Form Store 41g
41d
Object
User Control
Object Library

Figure 11

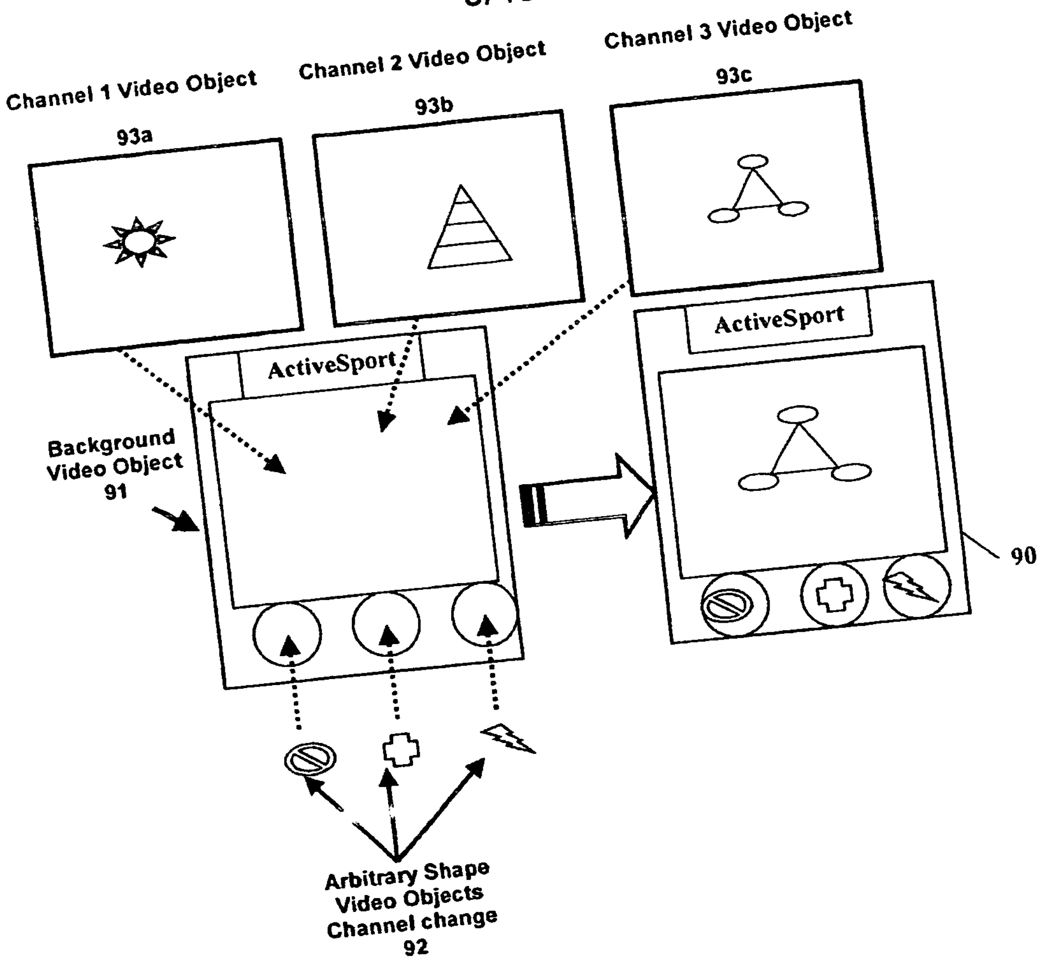

Channel 1 Video Object
93a
Channel 2 Video Object
93b
Channel 3 Video Object
93c
ActiveSport
ActiveSport
Background
Video Object
91
90
Arbitrary Shape
Video Objects
Channel change
92

Figure 12

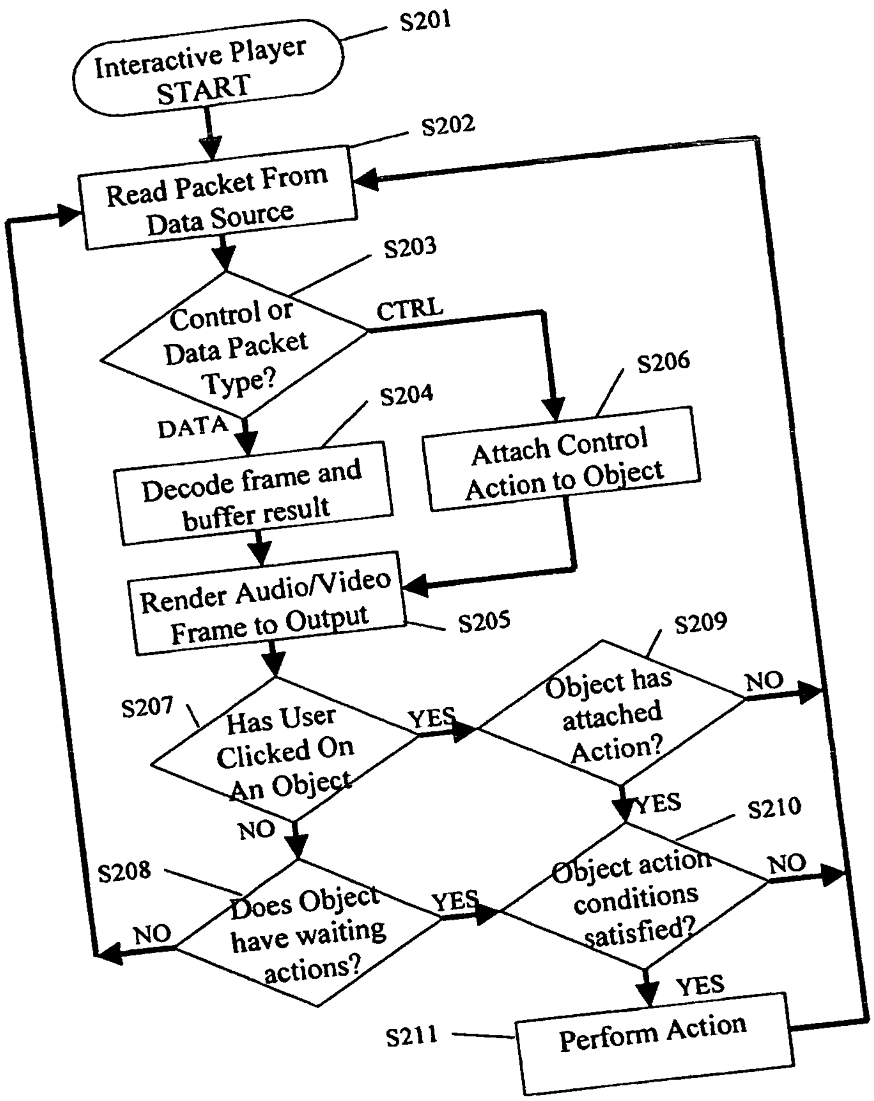

S201
Interactive Player START
S202
Read Packet From Data Source
S203
Control or Data Packet Type?
CTRL
DATA
S204
Decode frame and buffer result
S206
Attach Control Action to Object
Render Audio/Video Frame to Output
S205
S209
S207
Has User Clicked On An Object
YES
Object has attached Action?
NO
YES
S210
NO
S208
Does Object have waiting actions?
YES
Object action conditions satisfied?
NO
NO
YES
S211
Perform Action

Figure 13

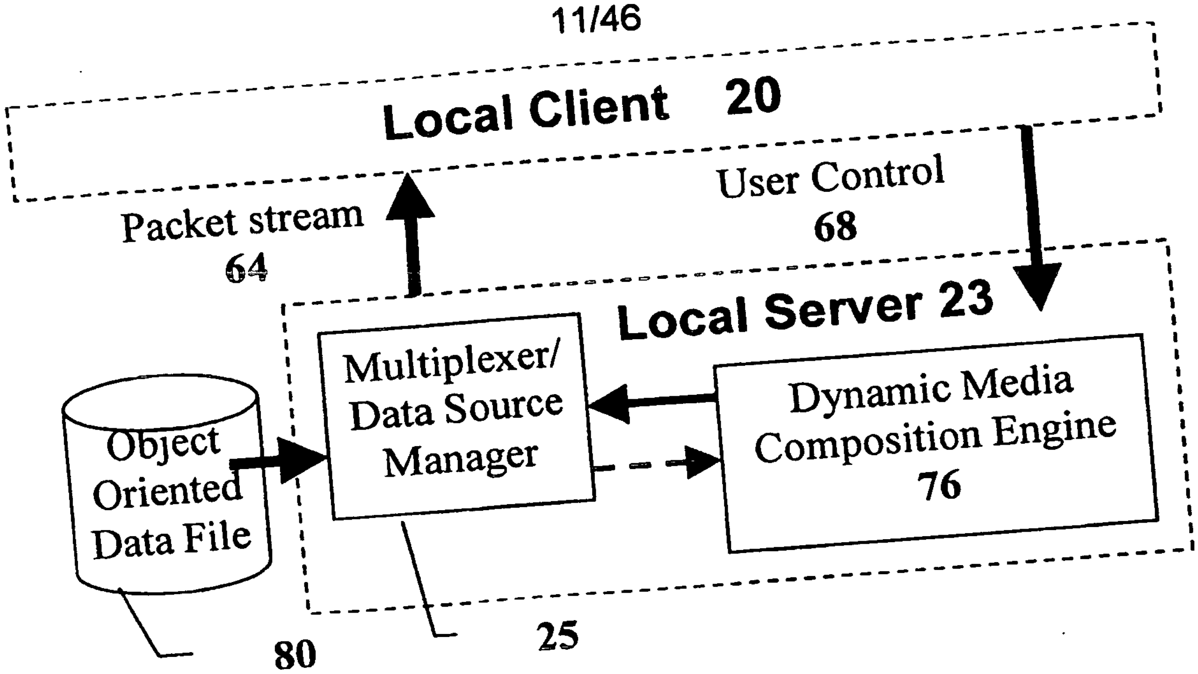

Local Client 20
Packet stream
64
User Control
68
Local Server 23
Multiplexer/
Data Source
Manager
Dynamic Media
Composition Engine
76
Object
Oriented
Data File
80
25

Figure 14

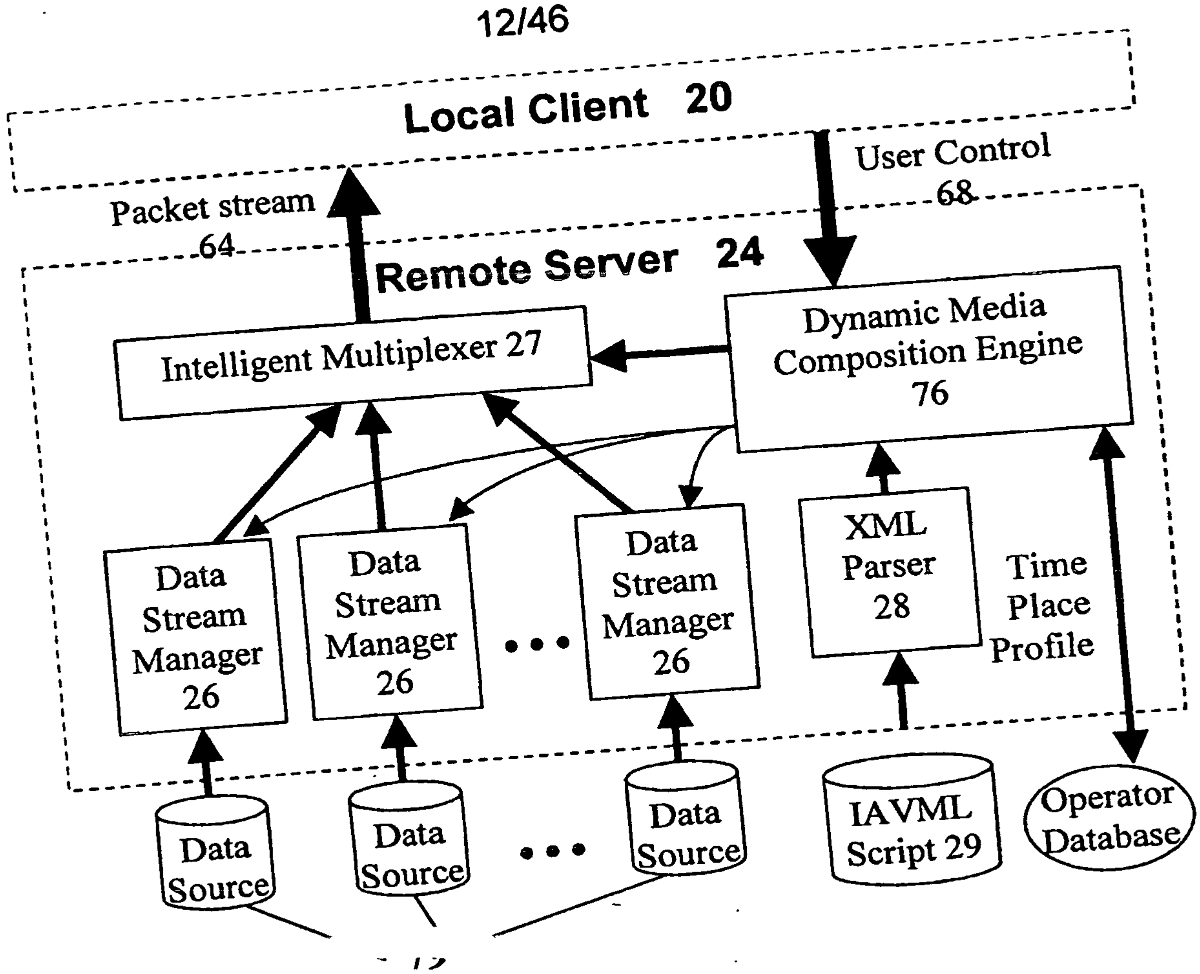

Local Client 20
User Control
68
Packet stream
64
Remote Server 24
Intelligent Multiplexer 27
Dynamic Media
Composition Engine
76
Data
Stream
Manager
26
Data
Stream
Manager
26
Data
Stream
Manager
26
XML
Parser
28
Time
Place
Profile
Data
Source
Data
Source
Data
Source
IAVML
Script 29
Operator
Database

Fig.

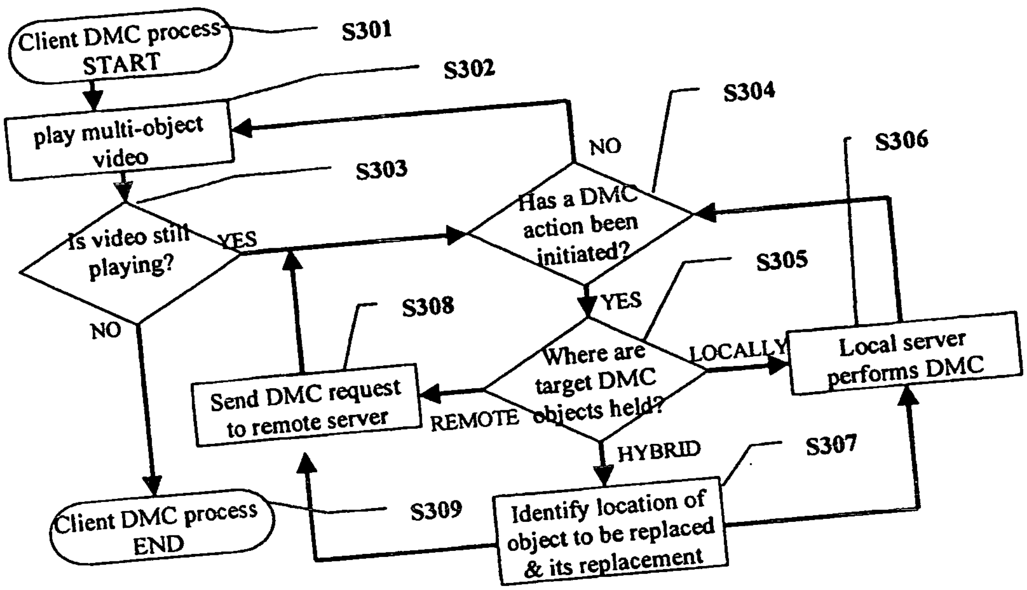

Client DMC process START
S301
play multi-object video
S302
S303
Is video still playing?
YES
NO
Has a DMC action been initiated?
NO
YES
S304
S305
S306
Where are target DMC objects held?
LOCALLY
REMOTE
HYBRID
Local server performs DMC
Send DMC request to remote server
S308
Identify location of object to be replaced & its replacement
S307
Client DMC process END
S309

Figure 16

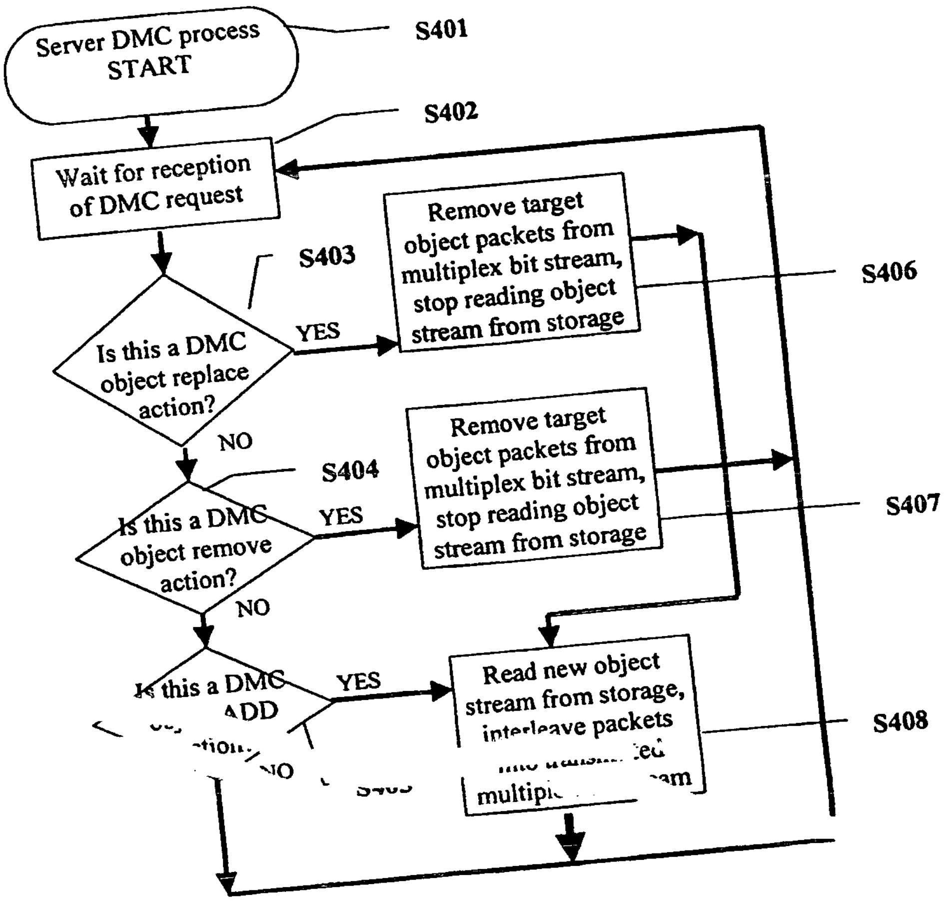

Server DMC process START
S401
Wait for reception of DMC request
S402
Is this a DMC object replace action?
S403
YES
NO
Remove target object packets from multiplex bit stream, stop reading object stream from storage
S406
Is this a DMC object remove action?
S404
YES
NO
Remove target object packets from multiplex bit stream, stop reading object stream from storage
S407
YES
NO
Read new object stream from storage, interleave packets
S408

Figure 17

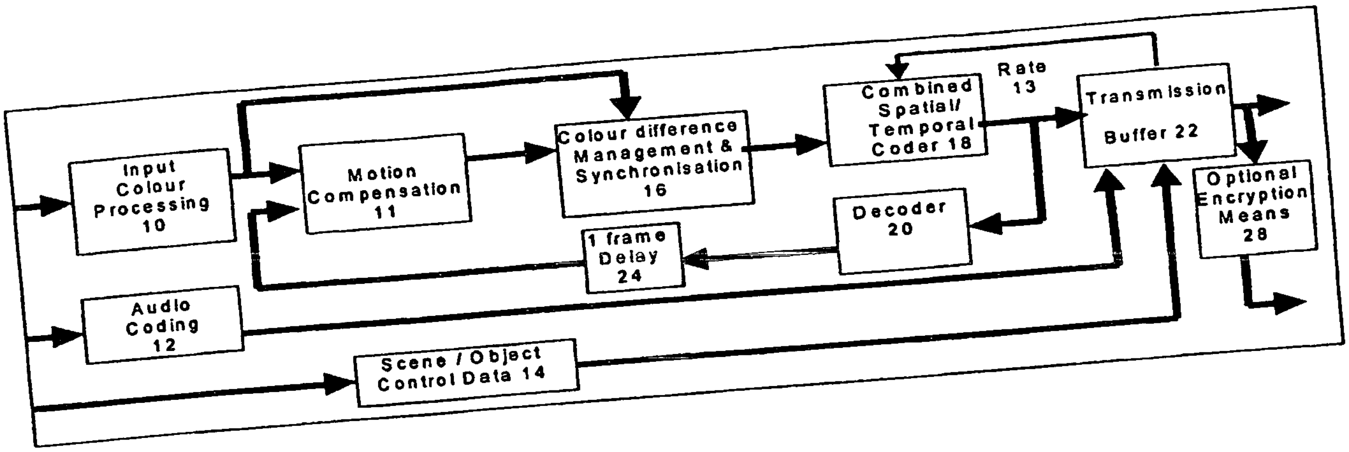

Input
Colour
Processing
10
Motion
Compensation
11
Colour difference
Management &
Synchronisation
16
Combined
Spatial/
Temporal
Coder 18
Rate
13
Transmission
Buffer 22
Optional
Encryption
Means
28
Decoder
20
1 frame
Delay
24
Audio
Coding
12
Scene / Object
Control Data 14

Figure 18

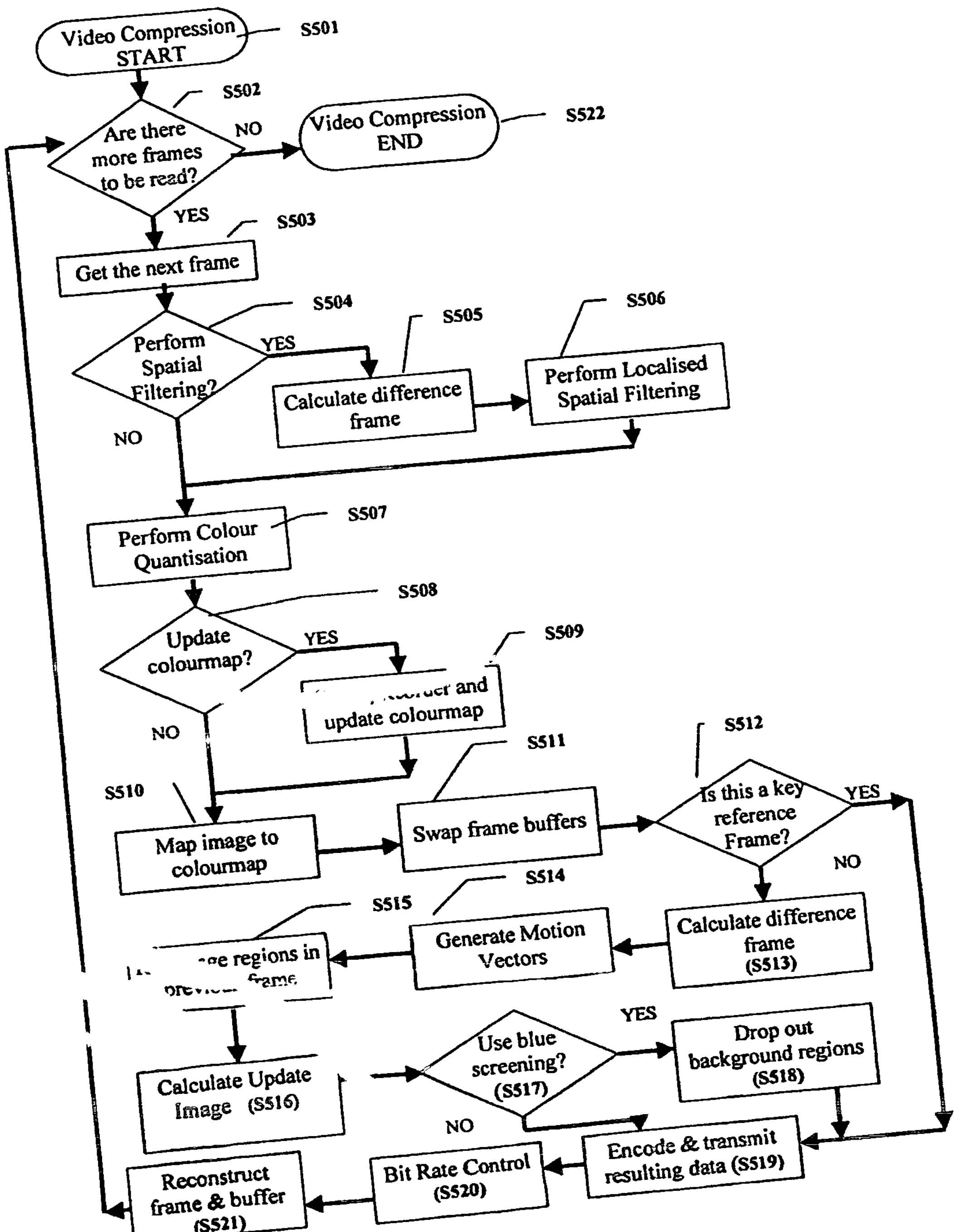

Video Compression START
S501
S502
Are there more frames to be read?
NO
Video Compression END
S522
YES
S503
Get the next frame
S504
Perform Spatial Filtering?
YES
S505
Calculate difference frame
S506
Perform Localised Spatial Filtering
NO
Perform Colour Quantisation
S507
S508
Update colourmap?
YES
S509
update colourmap
NO
S510
Map image to colourmap
S511
Swap frame buffers
S512
Is this a key reference Frame?
YES
NO
Calculate difference frame (S513)
S514
Generate Motion Vectors
S515
Calculate Update Image (S516)
Use blue screening? (S517)
YES
Drop out background regions (S518)
NO
Encode & transmit resulting data (S519)
Bit Rate Control (S520)
Reconstruct frame & buffer (S521)

Figure 19

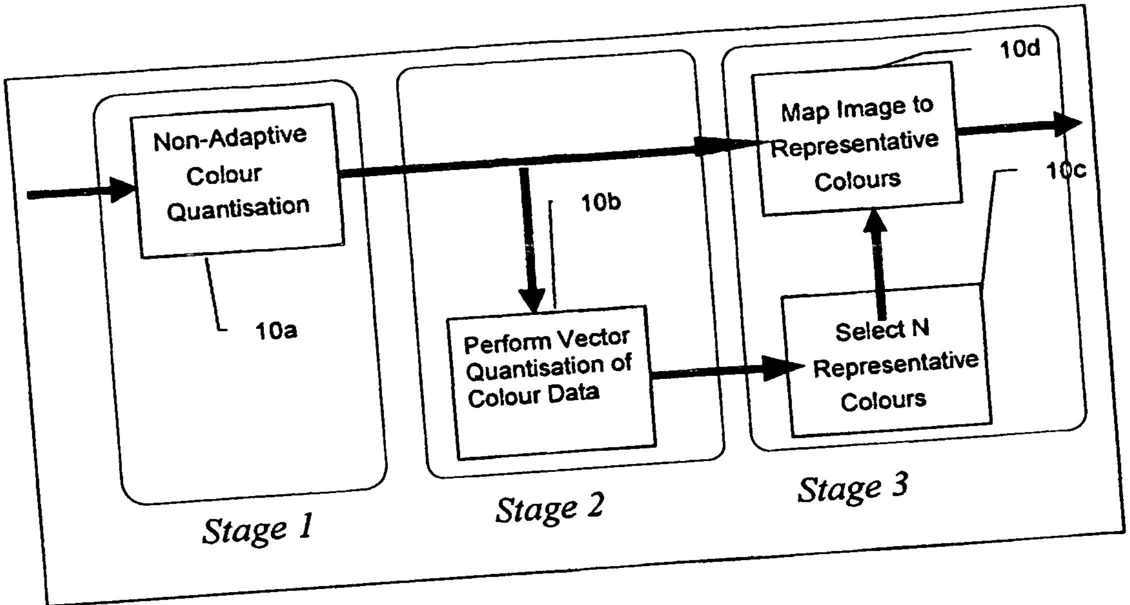


Figure 20

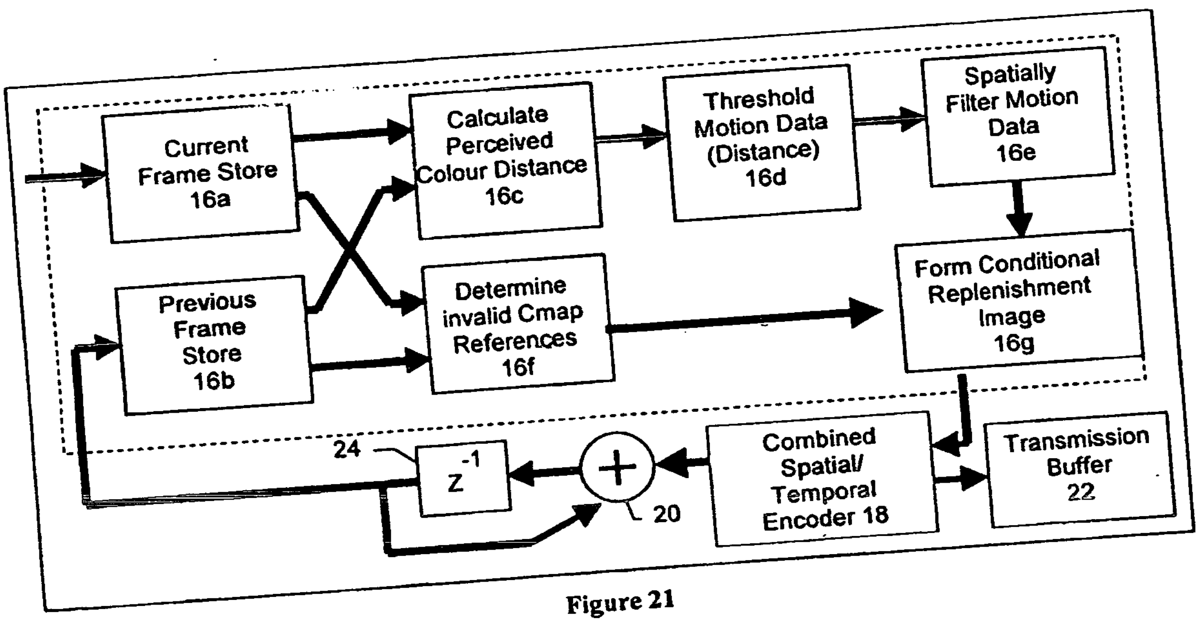


Figure 21

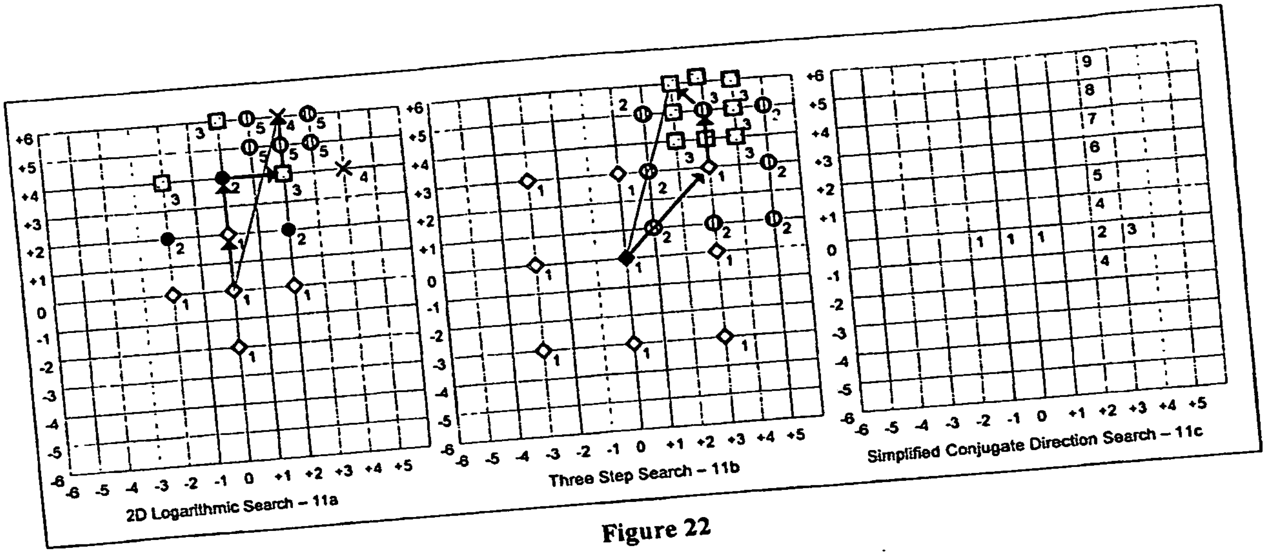


Figure 22

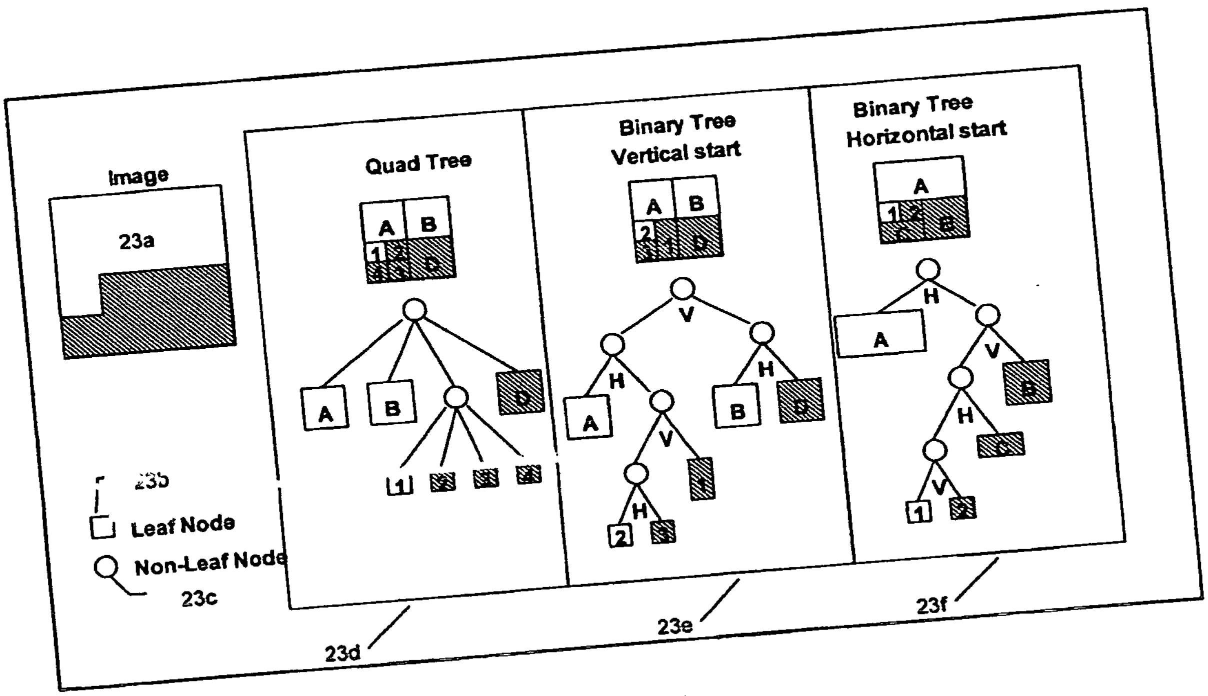


Figure 23

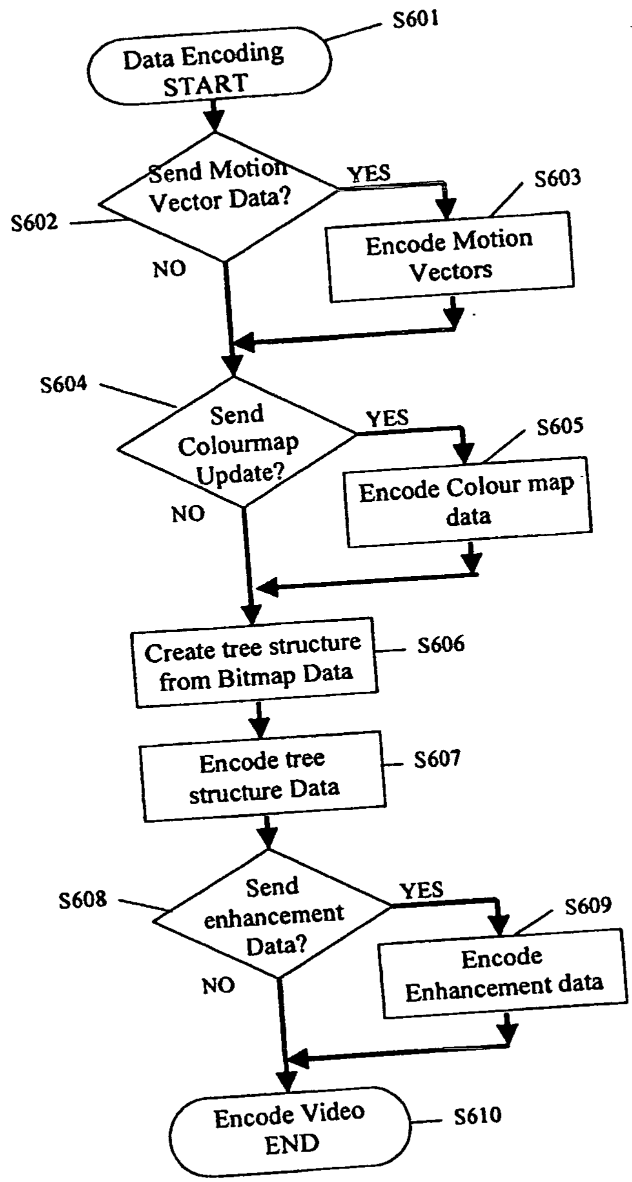

S601
Data Encoding START
Send Motion Vector Data?
S602
YES
NO
S603
Encode Motion Vectors
S604
Send Colourmap Update?
YES
NO
S605
Encode Colour map data
Create tree structure from Bitmap Data
S606
Encode tree structure Data
S607
S608
Send enhancement Data?
YES
NO
S609
Encode Enhancement data
Encode Video END
S610

Figure 24

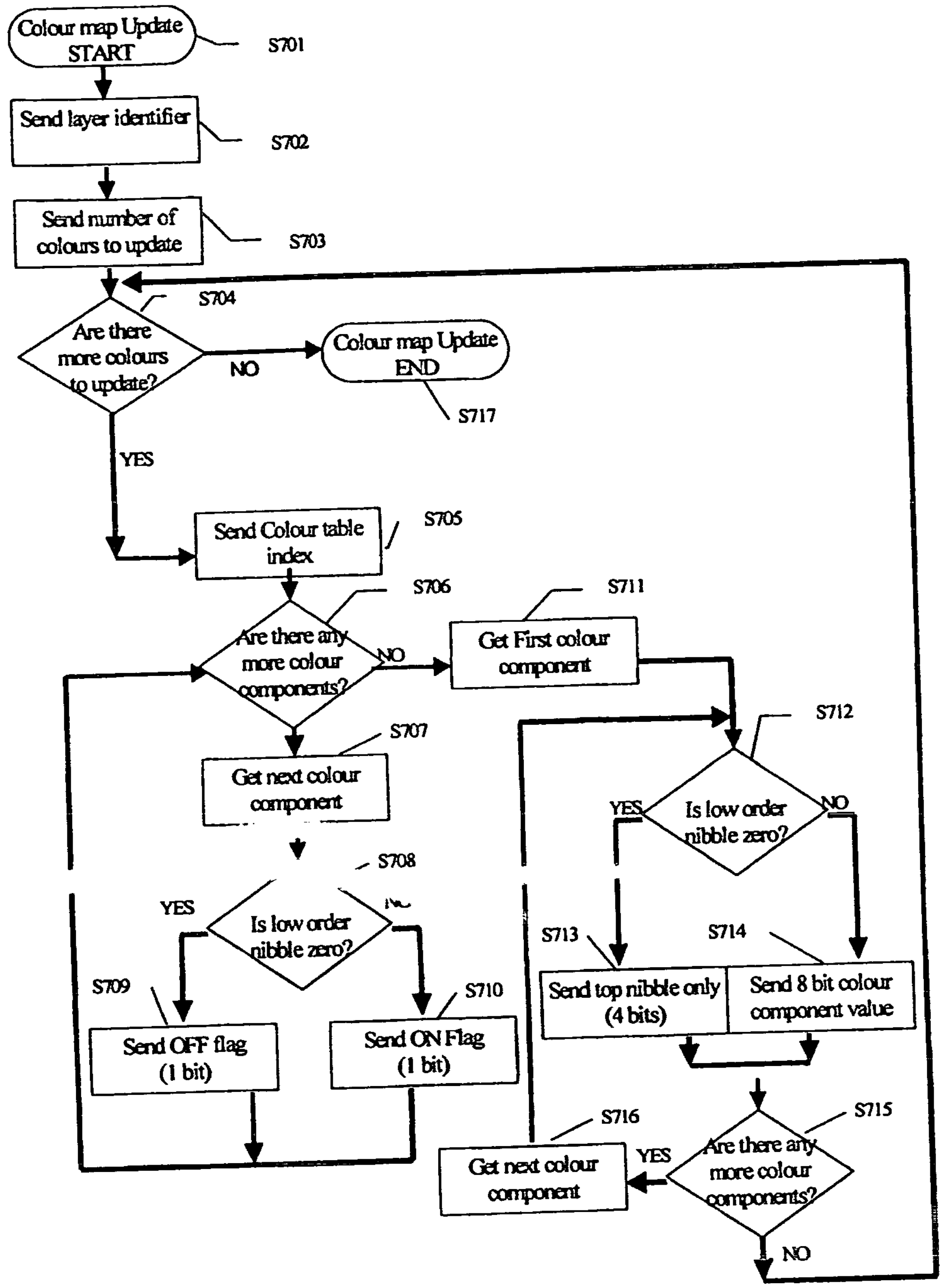

Colour map Update START
S701
Send layer identifier
S702
Send number of colours to update
S703
S704
Are there more colours to update?
NO
Colour map Update END
S717
YES
Send Colour table index
S705
S706
Are there any more colour components?
NO
Get First colour component
S711
S707
Get next colour component
S708
YES
Is low order nibble zero?
S709
Send OFF flag (1 bit)
S710
Send ON Flag (1 bit)
S712
YES
Is low order nibble zero?
NO
S713
S714
Send top nibble only (4 bits)
Send 8 bit colour component value
S716
S715
Get next colour component
YES
Are there any more colour components?
NO

Figure 25

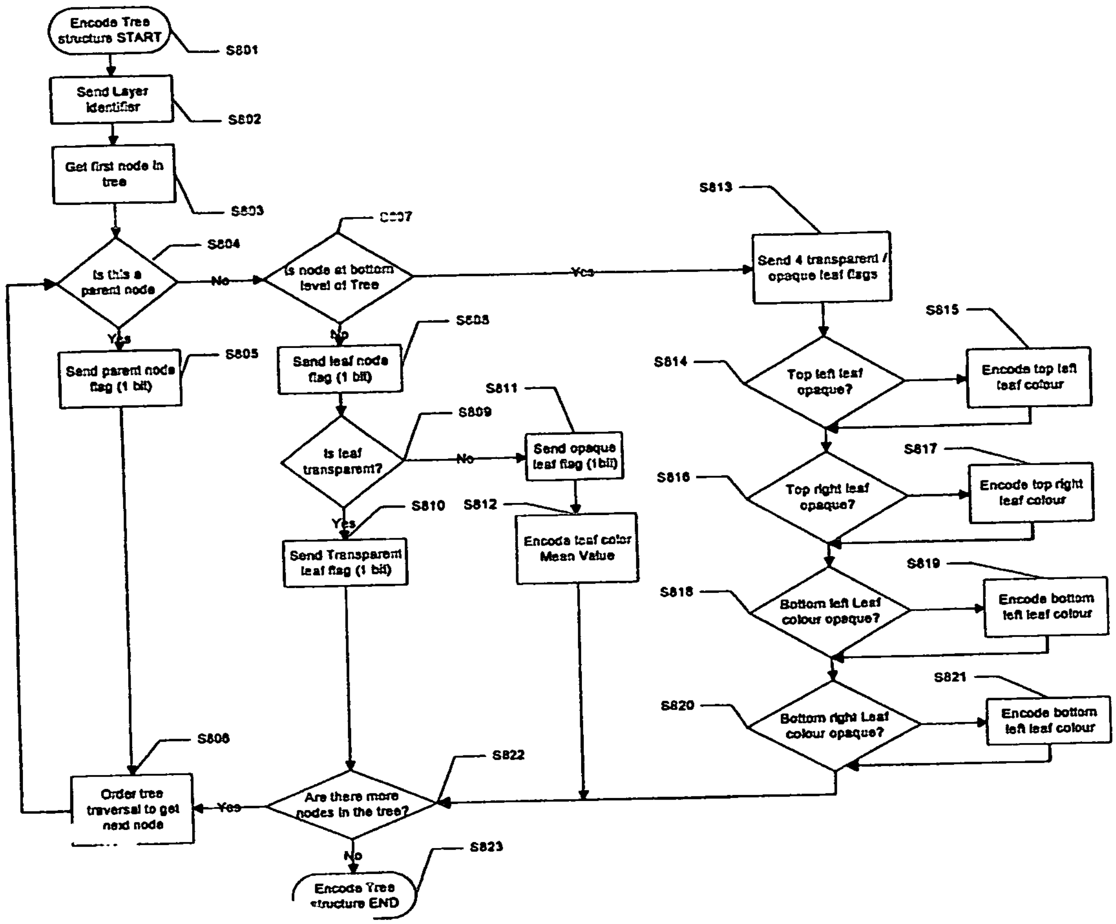

Encode Tree structure START
S801
Send Layer Identifier
S802
Get first node in tree
S803
Is this a parent node
S804
No
Yes
Send parent node flag (1 bit)
S805
Is node at bottom level of Tree
Yes
No
Send 4 transparent / opaque leaf flags
S813
Send leaf node flag (1 bit)
S808
Is leaf transparent?
S809
No
Yes
Send opaque leaf flag (1bit)
S811
Send Transparent leaf flag (1 bit)
S810
Encode leaf color Mean Value
S812
Top left leaf opaque?
S814
Encode top left leaf colour
S815
Top right leaf opaque?
S816
Encode top right leaf colour
S817
Bottom left Leaf colour opaque?
S818
Encode bottom left leaf colour
S819
Bottom right Leaf colour opaque?
S820
Encode bottom left leaf colour
S821
Order tree traversal to get next node
S806
Are there more nodes in the tree?
S822
Yes
No
Encode Tree structure END
S823

Figure 26

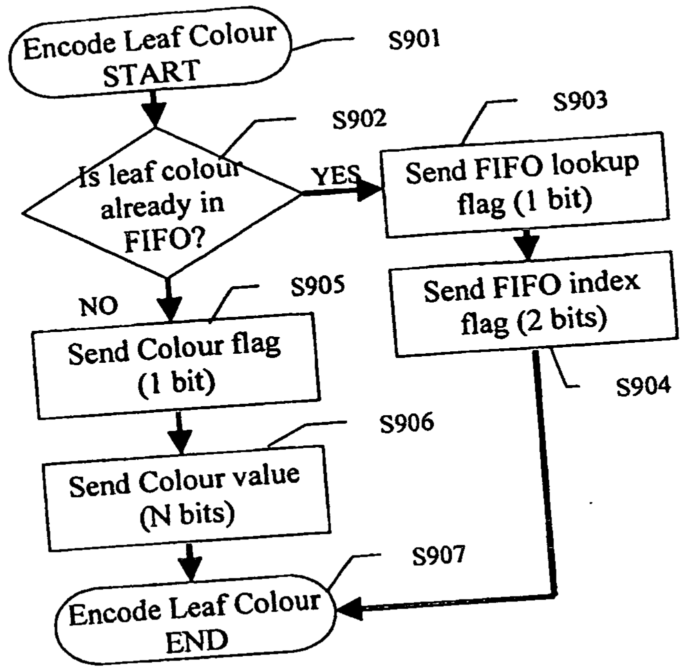

Encode Leaf Colour START
S901
Is leaf colour already in FIFO?
S902
YES
Send FIFO lookup flag (1 bit)
S903
Send FIFO index flag (2 bits)
S904
NO
Send Colour flag (1 bit)
S905
Send Colour value (N bits)
S906
Encode Leaf Colour END
S907

Figure 27

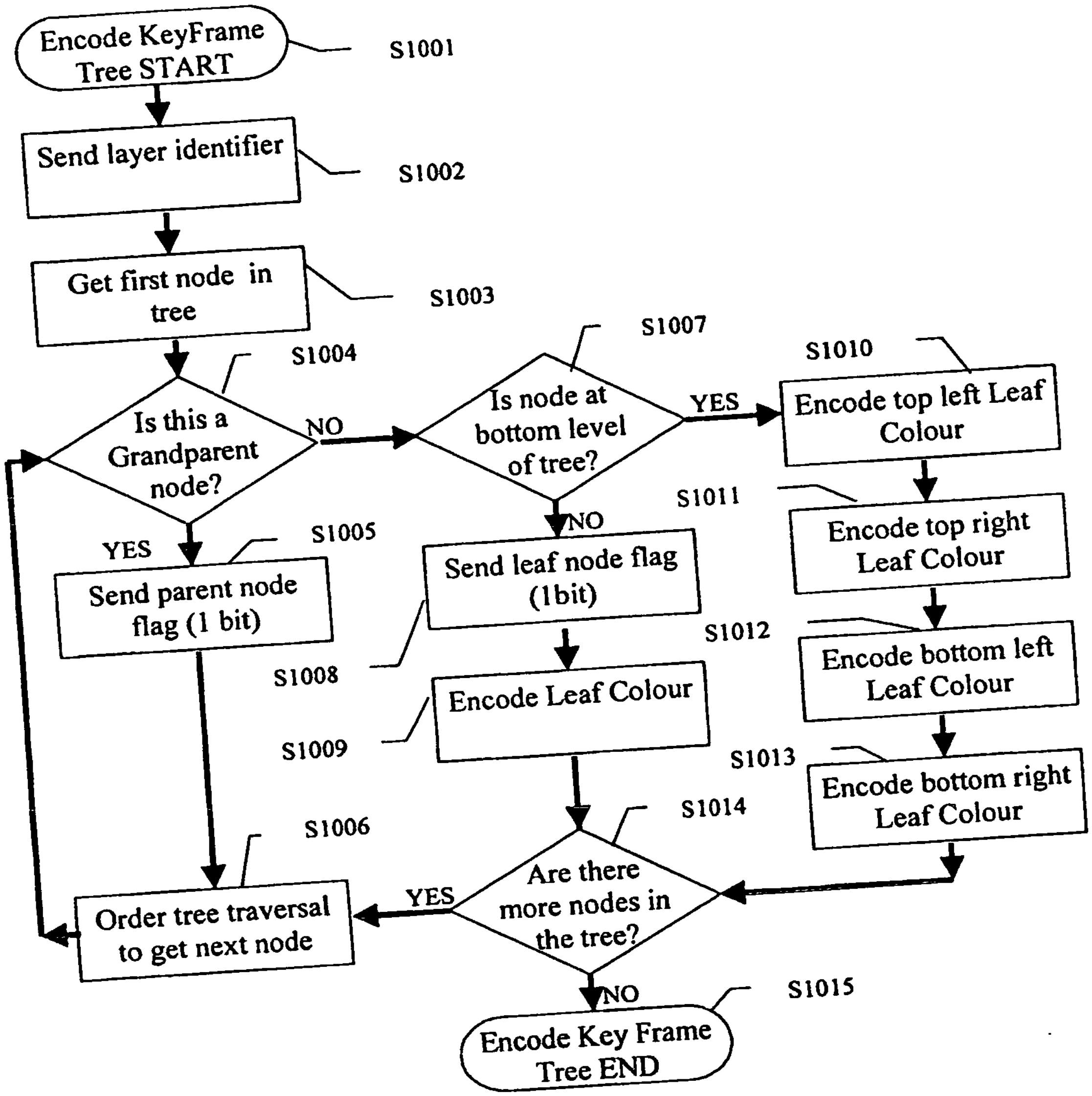
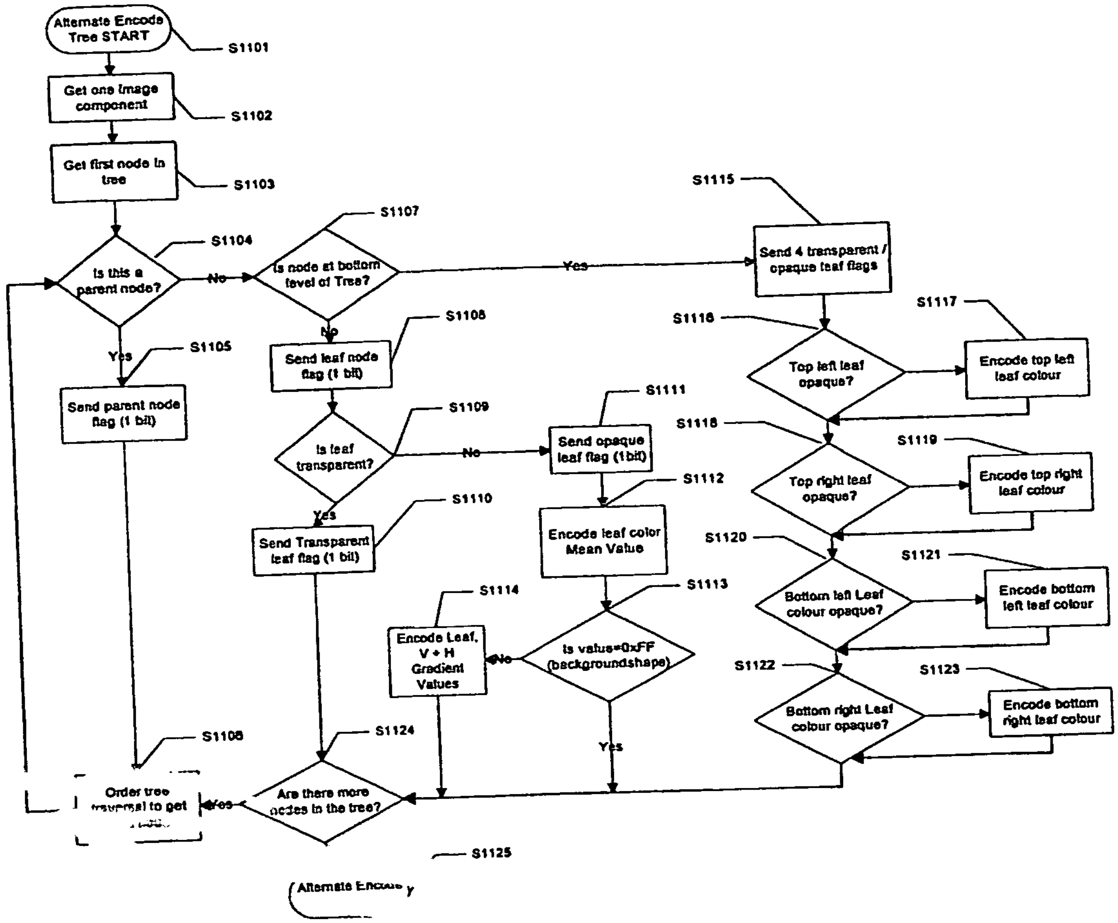

Encode KeyFrame Tree START
S1001
Send layer identifier
S1002
Get first node in tree
S1003
S1004
Is this a Grandparent node?
NO
YES
S1005
Send parent node flag (1 bit)
S1007
Is node at bottom level of tree?
YES
NO
S1010
Encode top left Leaf Colour
S1011
Encode top right Leaf Colour
S1012
Encode bottom left Leaf Colour
S1013
Encode bottom right Leaf Colour
Send leaf node flag (1bit)
S1008
Encode Leaf Colour
S1009
S1014
Are there more nodes in the tree?
YES
NO
S1006
Order tree traversal to get next node
S1015
Encode Key Frame Tree END

Figure 28


Alternate Encode Tree START
S1101
Get one Image component
S1102
Get first node in tree
S1103
Is this a parent node?
S1104
No
Yes
S1105
Send parent node flag (1 bit)
Is node at bottom level of Tree?
S1107
Yes
No
Send leaf node flag (1 bit)
S1108
Is leaf transparent?
S1109
No
Yes
Send Transparent leaf flag (1 bit)
S1110
Send opaque leaf flag (1bit)
S1111
Encode leaf color Mean Value
S1112
Is value=0xFF (backgroundshape)
S1113
No
Yes
Encode Leaf, V + H Gradient Values
S1114
Send 4 transparent / opaque leaf flags
S1115
Top left leaf opaque?
S1116
Encode top left leaf colour
S1117
Top right leaf opaque?
S1118
Encode top right leaf colour
S1119
Bottom left Leaf colour opaque?
S1120
Encode bottom left leaf colour
S1121
Bottom right Leaf colour opaque?
S1122
Encode bottom right leaf colour
S1123
Are there more nodes in the tree?
S1124
Yes
Order tree traversal to get
S1106
S1125
Alternate Encode

...re 29

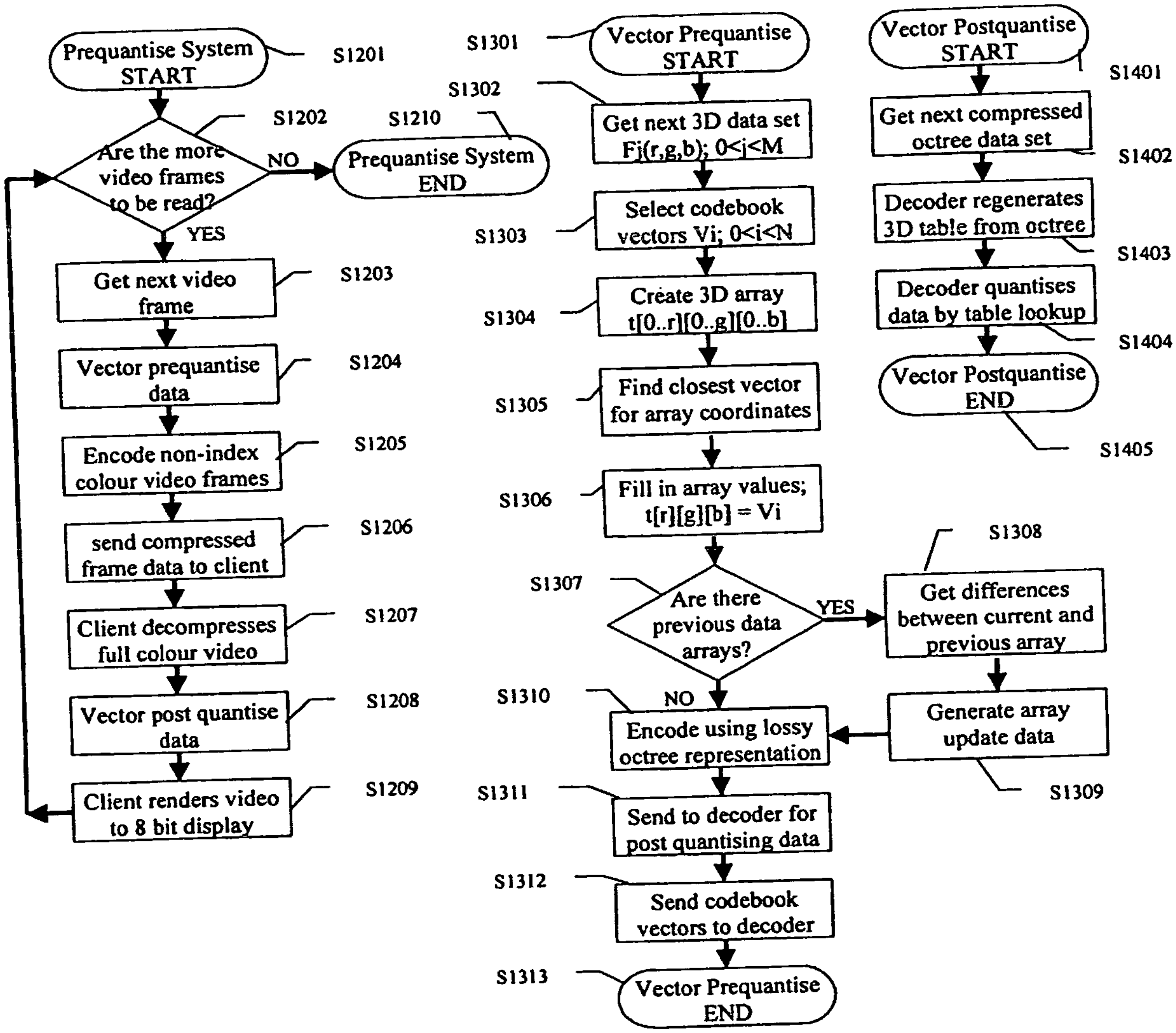

Prequantise System START
S1201
Are the more video frames to be read?
S1202
NO
YES
Prequantise System END
S1210
Get next video frame
S1203
Vector prequantise data
S1204
Encode non-index colour video frames
S1205
send compressed frame data to client
S1206
Client decompresses full colour video
S1207
Vector post quantise data
S1208
Client renders video to 8 bit display
S1209
Vector Prequantise START
S1301
Get next 3D data set Fj(r,g,b); 0<j<M
S1302
Select codebook vectors Vi; 0<i<N
S1303
Create 3D array t[0..r][0..g][0..b]
S1304
Find closest vector for array coordinates
S1305
Fill in array values; t[r][g][b] = Vi
S1306
Are there previous data arrays?
S1307
YES
NO
Get differences between current and previous array
S1308
Generate array update data
S1309
Encode using lossy octree representation
S1310
Send to decoder for post quantising data
S1311
Send codebook vectors to decoder
S1312
Vector Prequantise END
S1313
Vector Postquantise START
S1401
Get next compressed octree data set
S1402
Decoder regenerates 3D table from octree
S1403
Decoder quantises data by table lookup
S1404
Vector Postquantise END
S1405

Figure 30

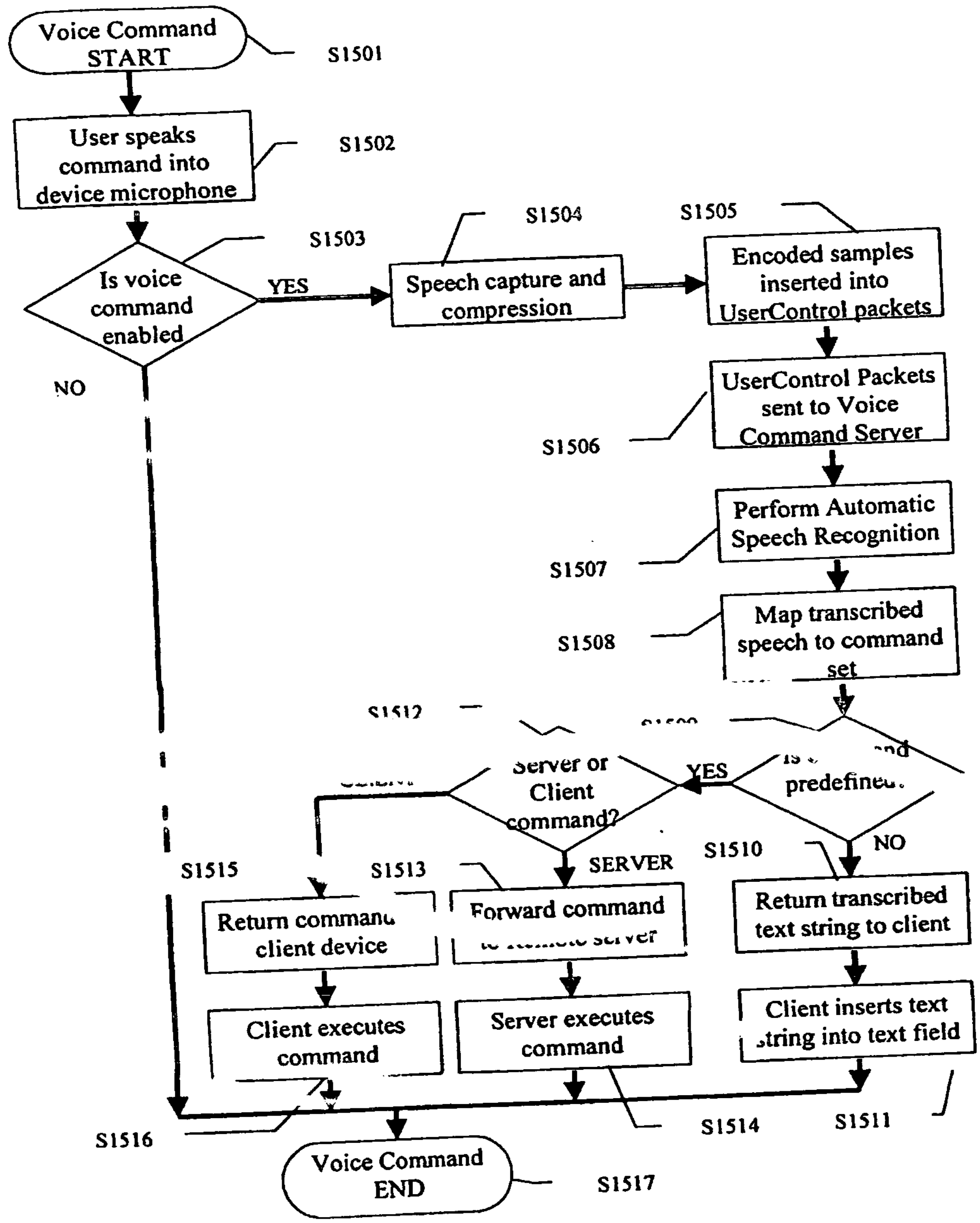
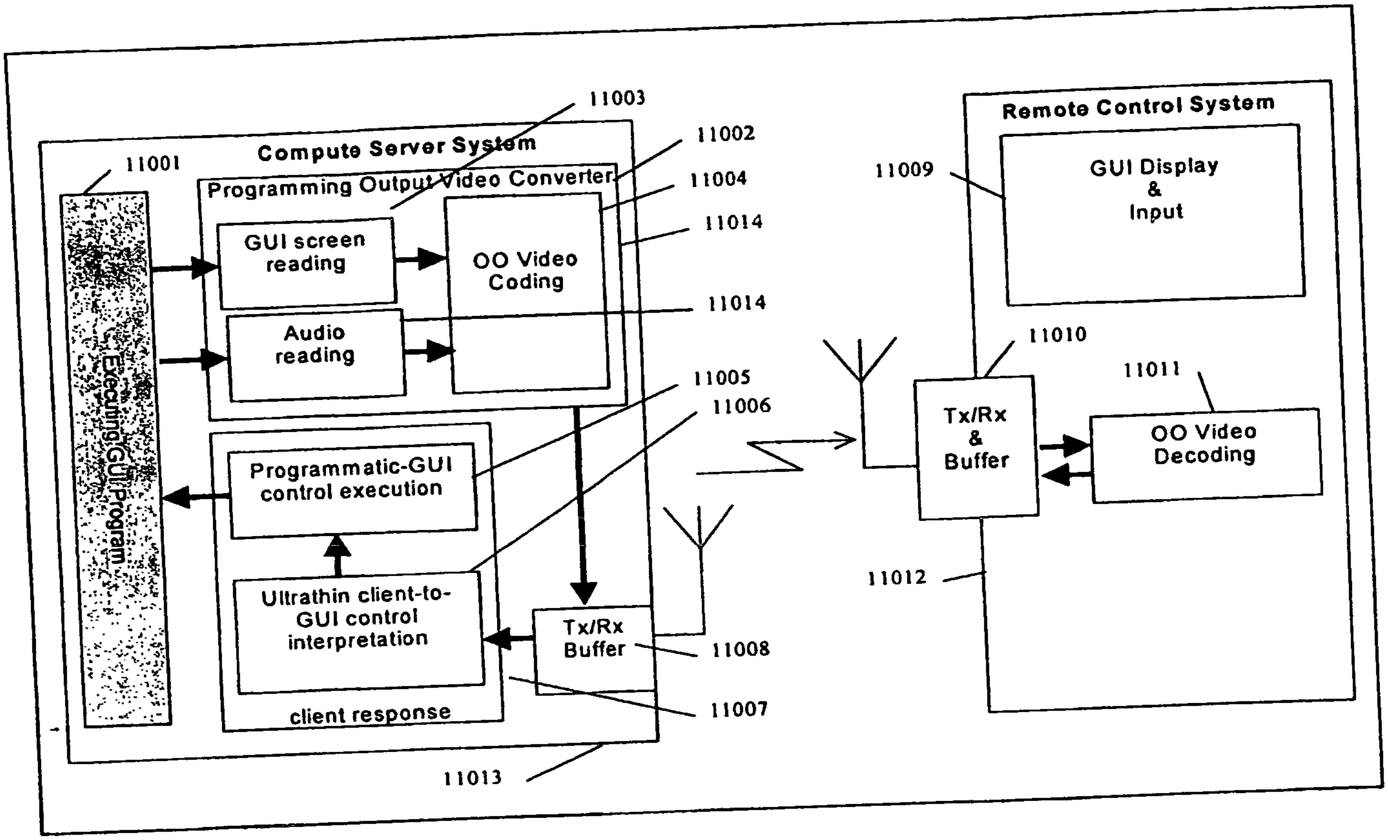
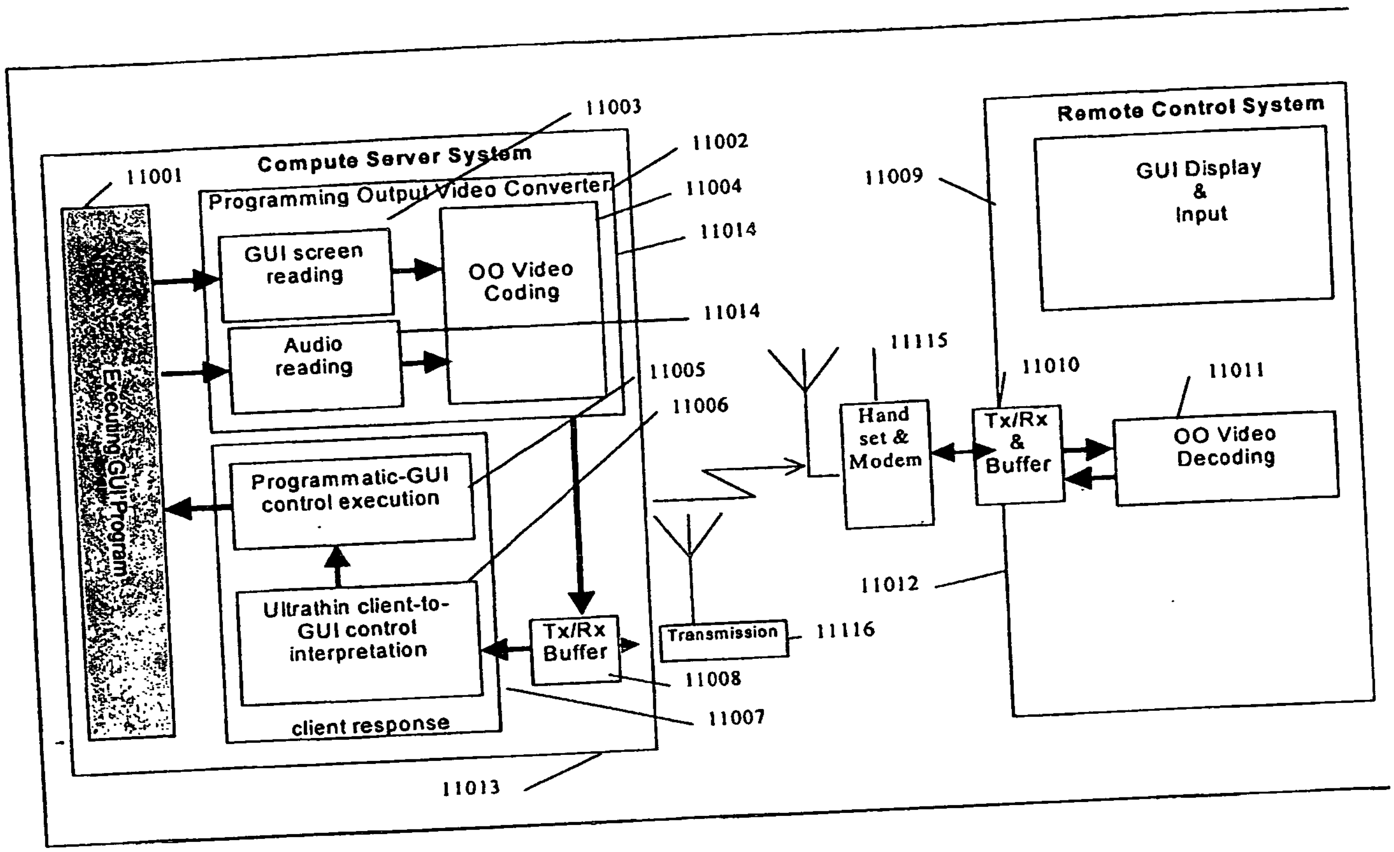

Voice Command START
S1501
User speaks command into device microphone
S1502
S1503
Is voice command enabled
YES
NO
S1504
Speech capture and compression
S1505
Encoded samples inserted into UserControl packets
UserControl Packets sent to Voice Command Server
S1506
Perform Automatic Speech Recognition
S1507
Map transcribed speech to command set
S1508
Client
Server or Client command?
YES
predefined
NO
S1515
S1513
SERVER
S1510
Return command client device
Forward command
Return transcribed text string to client
Client executes command
Server executes command
Client inserts text string into text field
S1516
S1514
S1511
Voice Command END
S1517

Figure 31

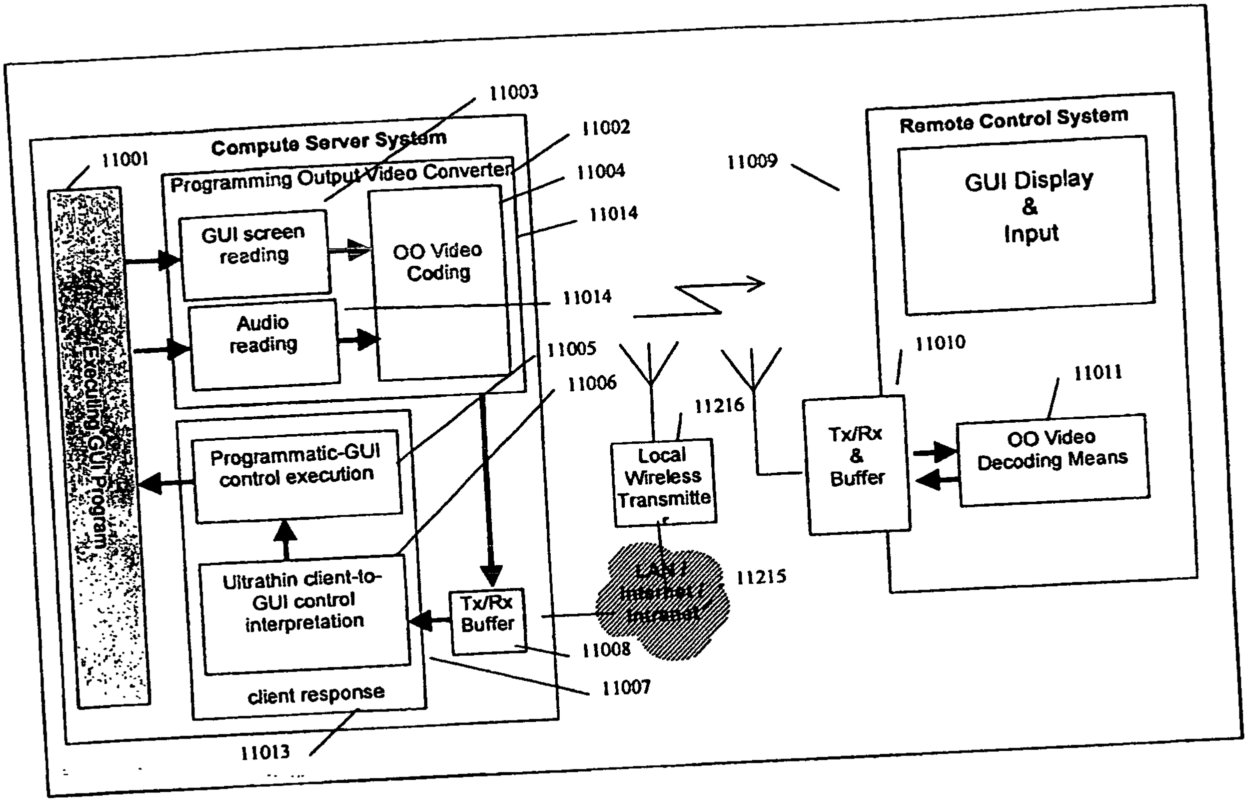

Compute Server System
Programming Output Video Converter
GUI screen reading
Audio reading
OO Video Coding
Executing GUI Program
Programmatic-GUI control execution
Ultrathin client-to-GUI control interpretation
client response
Tx/Rx Buffer
Remote Control System
GUI Display & Input
Tx/Rx & Buffer
OO Video Decoding
11001
11002
11003
11004
11014
11014
11005
11006
11007
11008
11009
11010
11011
11012
11013

Fi[illegible]re 32



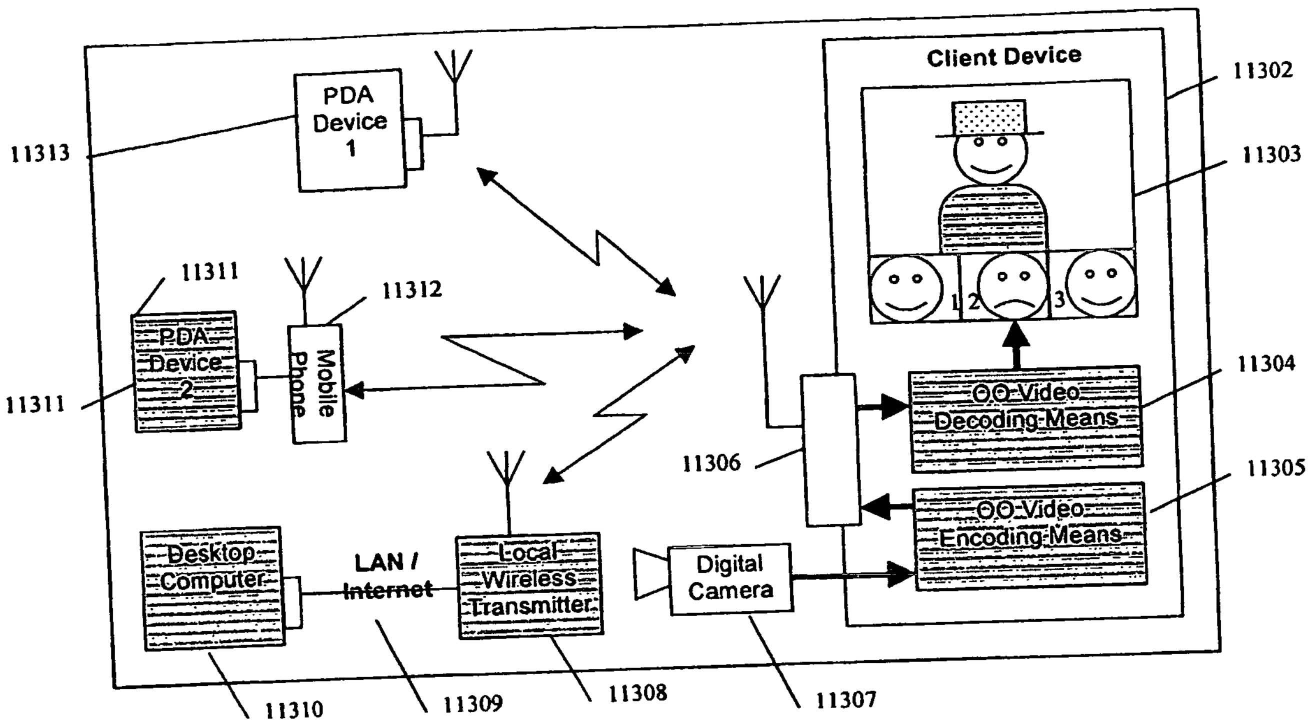
Compute Server System
Programming Output Video Converter
GUI screen reading
OO Video Coding
Audio reading
Executing GUI Program
Programmatic-GUI control execution
Ultrathin client-to-GUI control interpretation
client response
Tx/Rx Buffer
Transmission
Hand set & Modem
Tx/Rx & Buffer
OO Video Decoding
Remote Control System
GUI Display & Input
11001
11002
11003
11004
11005
11006
11007
11008
11009
11010
11011
11012
11013
11014
11014
11115
11116

Figure 33



Compute Server System
Programming Output Video Converter
GUI screen reading
OO Video Coding
Audio reading
Programmatic-GUI control execution
Ultrathin client-to-GUI control interpretation
client response
Tx/Rx Buffer
Local Wireless Transmitter
LAN / Internet / Intranet
Remote Control System
GUI Display & Input
Tx/Rx & Buffer
OO Video Decoding Means
11001
11002
11003
11004
11005
11006
11007
11008
11009
11010
11011
11013
11014
11014
11215
11216

Figure 34


Client Device
11302
11303
11313
PDA
Device
1
11311
11312
PDA
Device
2
Mobile
Phone
11311
11304
OO Video
Decoding Means
11306
11305
OO Video
Encoding Means
Desktop
Computer
LAN /
Internet
Local
Wireless
Transmitter
Digital
Camera
11310
11309
11308
11307
1
2
3

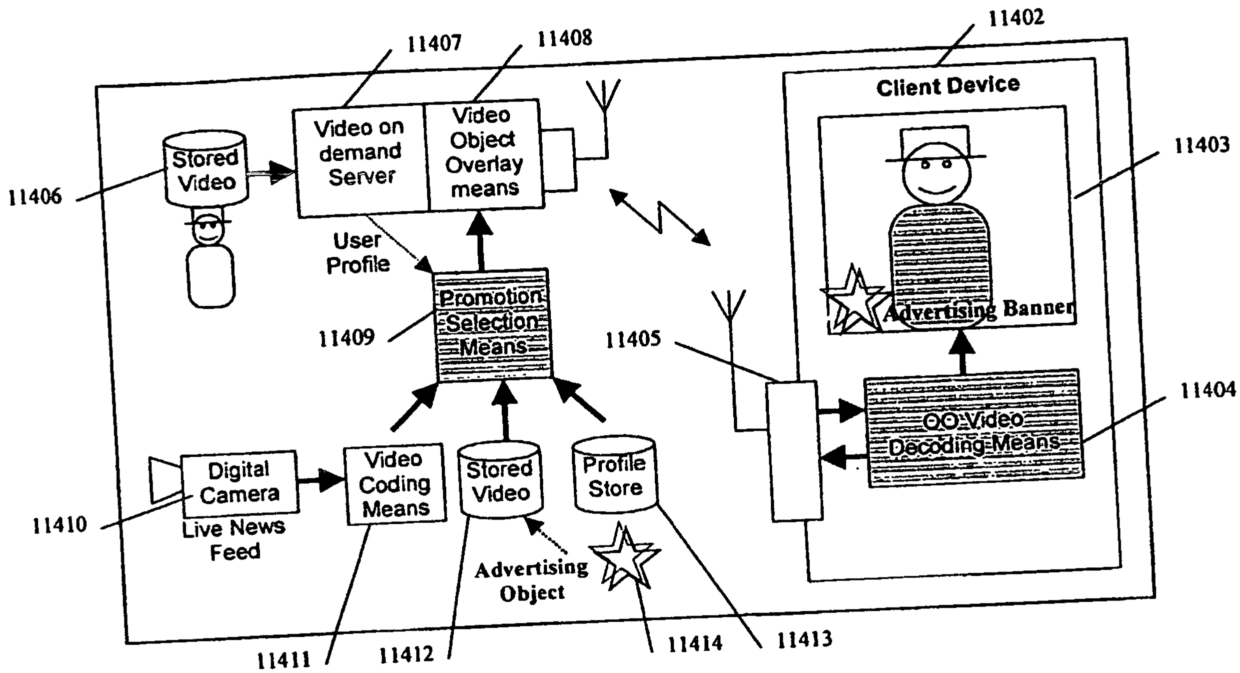
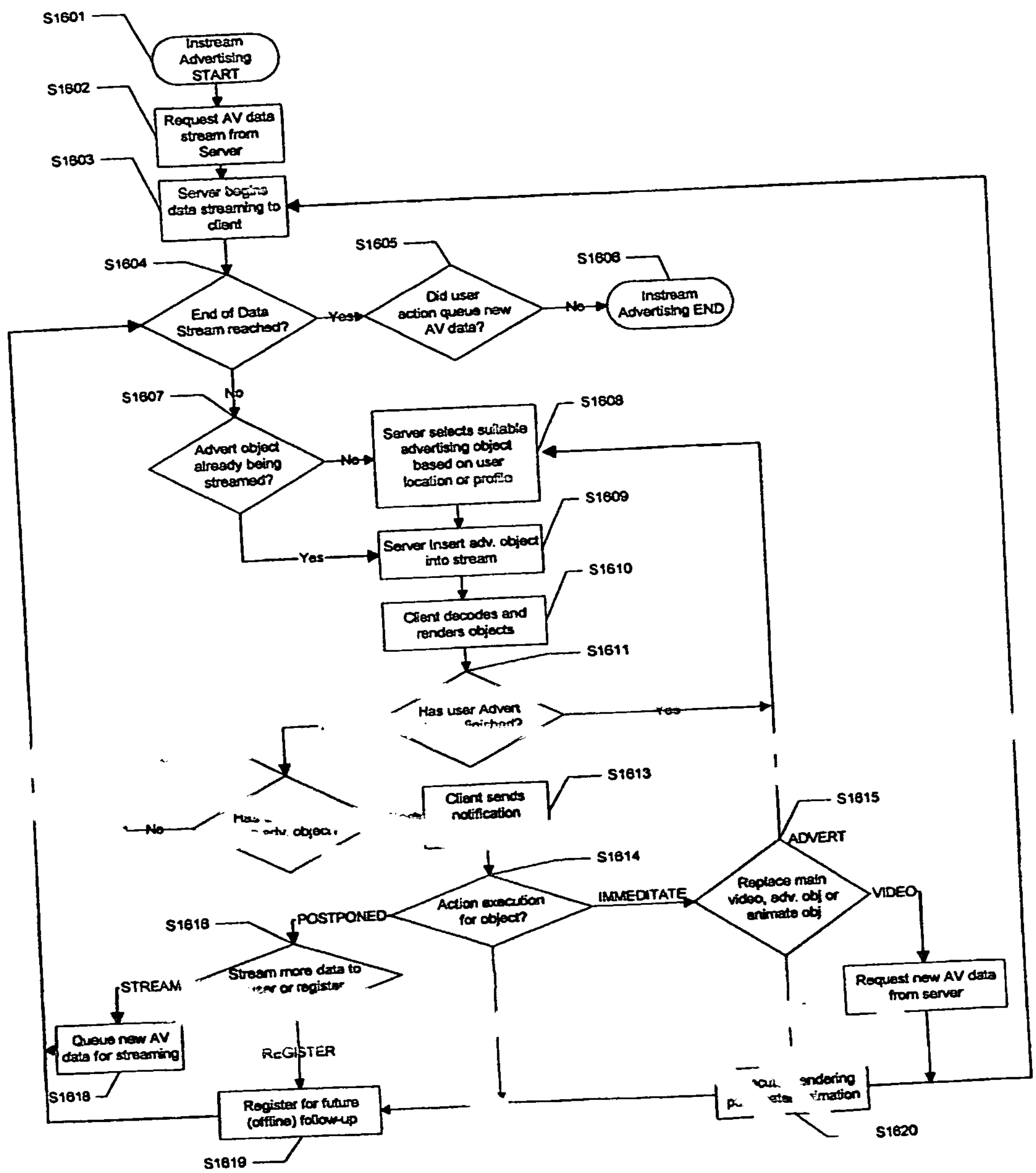

11407
11408
11402
Client Device
11403
Stored Video
Video on demand Server
Video Object Overlay means
11406
User Profile
Promotion Selection Means
11409
11405
Advertising Banner
11404
OO Video Decoding Means
Digital Camera
Video Coding Means
Stored Video
Profile Store
11410
Live News Feed
Advertising Object
11411
11412
11414
11413

Figure 36


S1601
Instream Advertising START
S1602
Request AV data stream from Server
S1603
Server begins data streaming to client
S1604
End of Data Stream reached?
Yes
S1605
Did user action queue new AV data?
No
S1606
Instream Advertising END
No
S1607
Advert object already being streamed?
No
S1608
Server selects suitable advertising object based on user location or profile
Yes
S1609
Server insert adv. object into stream
S1610
Client decodes and renders objects
S1611
Has user Advert
Yes
No
S1613
Client sends notification
S1614
Action execution for object?
IMMEDIATE
POSTPONED
S1615
ADVERT
Replace main video, adv. obj or animate obj
VIDEO
Request new AV data from server
S1616
Stream more data to
STREAM
Queue new AV data for streaming
S1618
REGISTER
Register for future (offline) follow-up
S1619
S1620

Figure

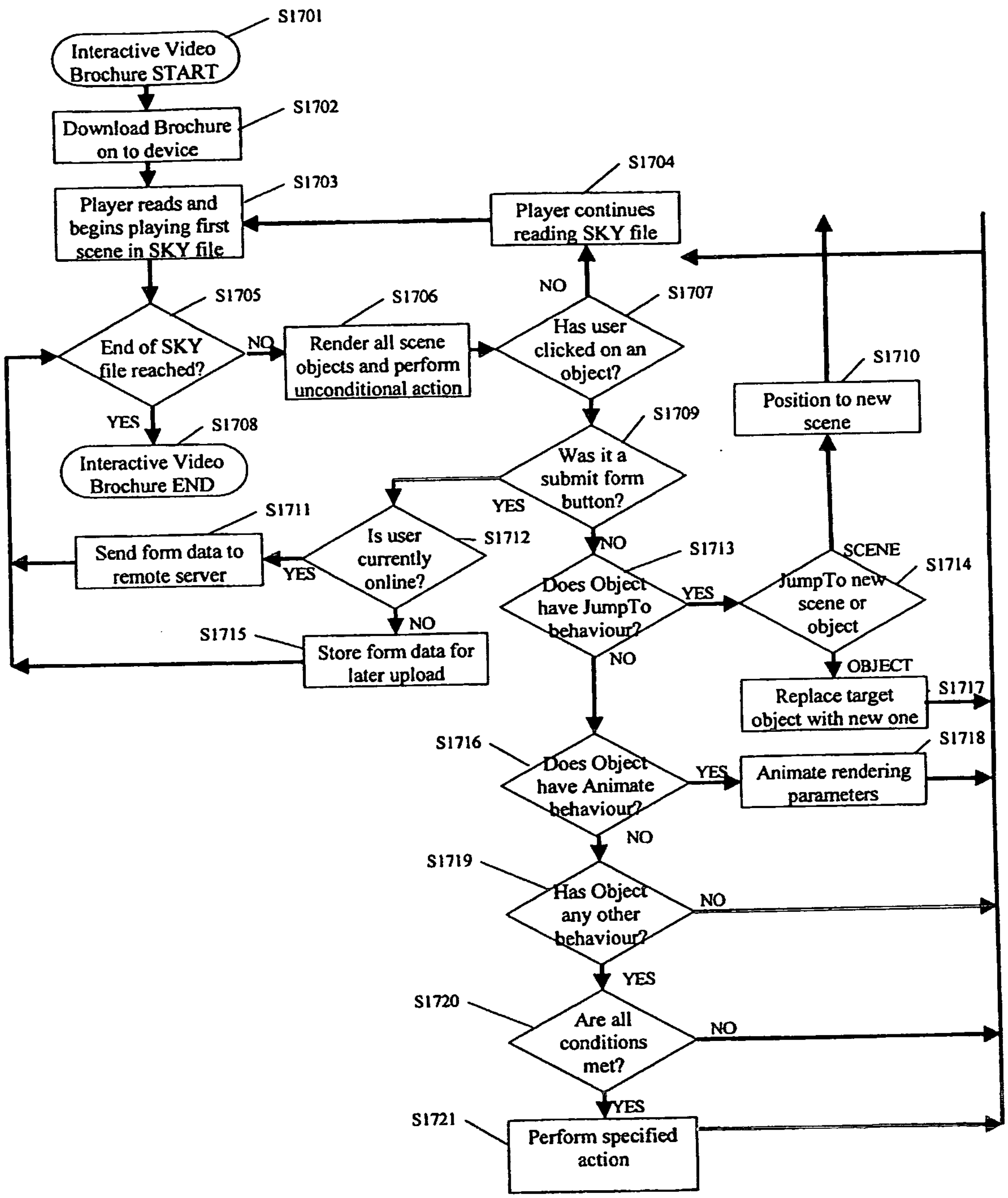

S1701
Interactive Video Brochure START
S1702
Download Brochure on to device
S1703
Player reads and begins playing first scene in SKY file
S1704
Player continues reading SKY file
S1705
End of SKY file reached?
NO
S1706
Render all scene objects and perform unconditional action
S1707
Has user clicked on an object?
NO
YES
S1708
Interactive Video Brochure END
S1709
Was it a submit form button?
YES
NO
S1710
Position to new scene
S1711
Send form data to remote server
S1712
Is user currently online?
YES
NO
S1713
Does Object have JumpTo behaviour?
YES
NO
S1714
JumpTo new scene or object
SCENE
OBJECT
S1715
Store form data for later upload
S1716
Does Object have Animate behaviour?
YES
NO
S1717
Replace target object with new one
S1718
Animate rendering parameters
S1719
Has Object any other behaviour?
NO
YES
S1720
Are all conditions met?
NO
YES
S1721
Perform specified action

Figure 38

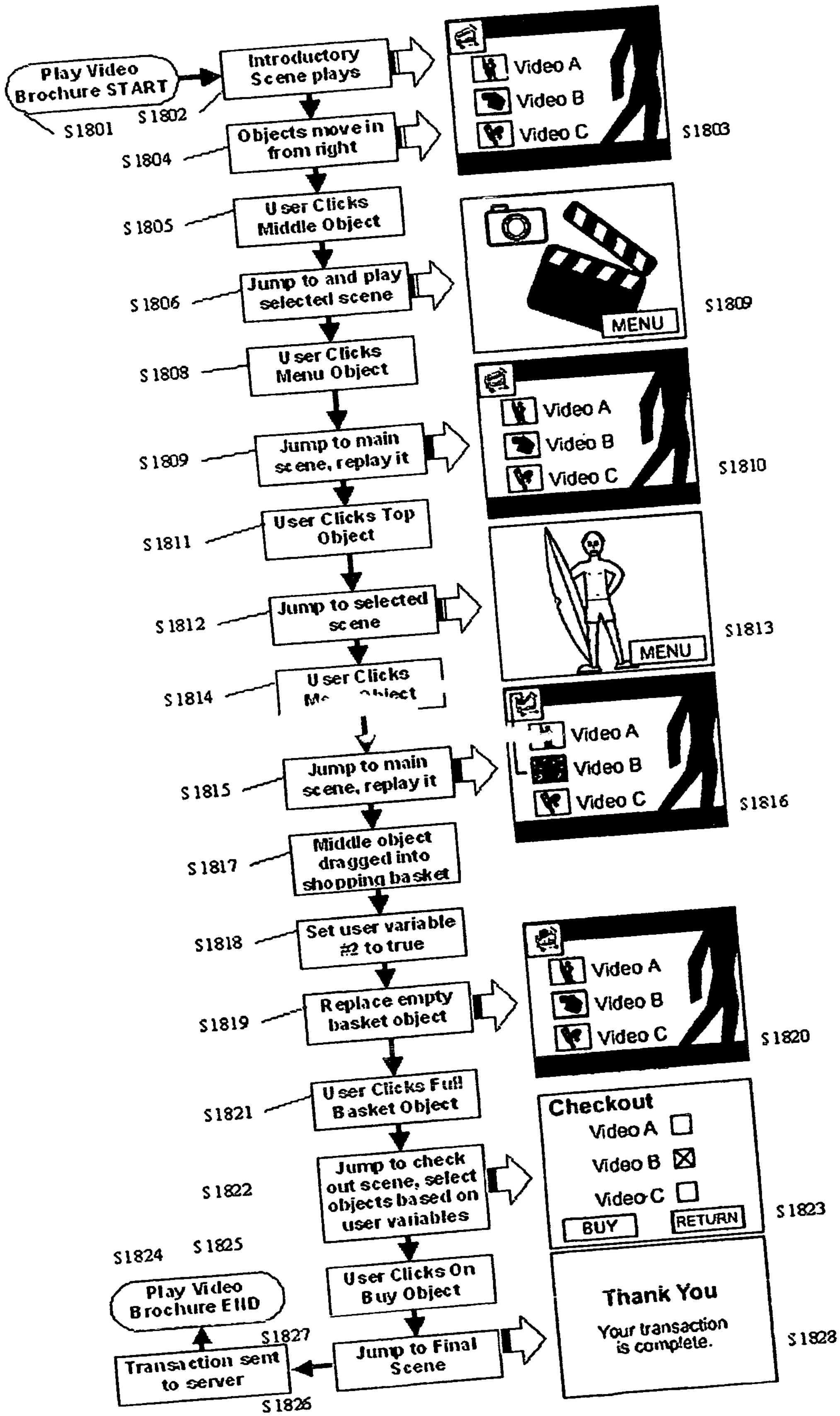

Play Video Brochure START
S1801
Introductory Scene plays
S1802
Objects move in from right
S1804
Video A
Video B
Video C
S1803
User Clicks Middle Object
S1805
Jump to and play selected scene
S1806
MENU
S1809
User Clicks Menu Object
S1808
Jump to main scene, replay it
S1809
Video A
Video B
Video C
S1810
User Clicks Top Object
S1811
Jump to selected scene
S1812
MENU
S1813
User Clicks
S1814
Jump to main scene, replay it
S1815
Video A
Video B
Video C
S1816
Middle object dragged into shopping basket
S1817
Set user variable #2 to true
S1818
Replace empty basket object
S1819
Video A
Video B
Video C
S1820
User Clicks Full Basket Object
S1821
Jump to check out scene, select objects based on user variables
S1822
Checkout
Video A
Video B
Video C
BUY
RETURN
S1823
S1824
S1825
Play Video Brochure END
User Clicks On Buy Object
S1827
Transaction sent to server
S1826
Jump to Final Scene
Thank You
Your transaction is complete.
S1828

Figure 39

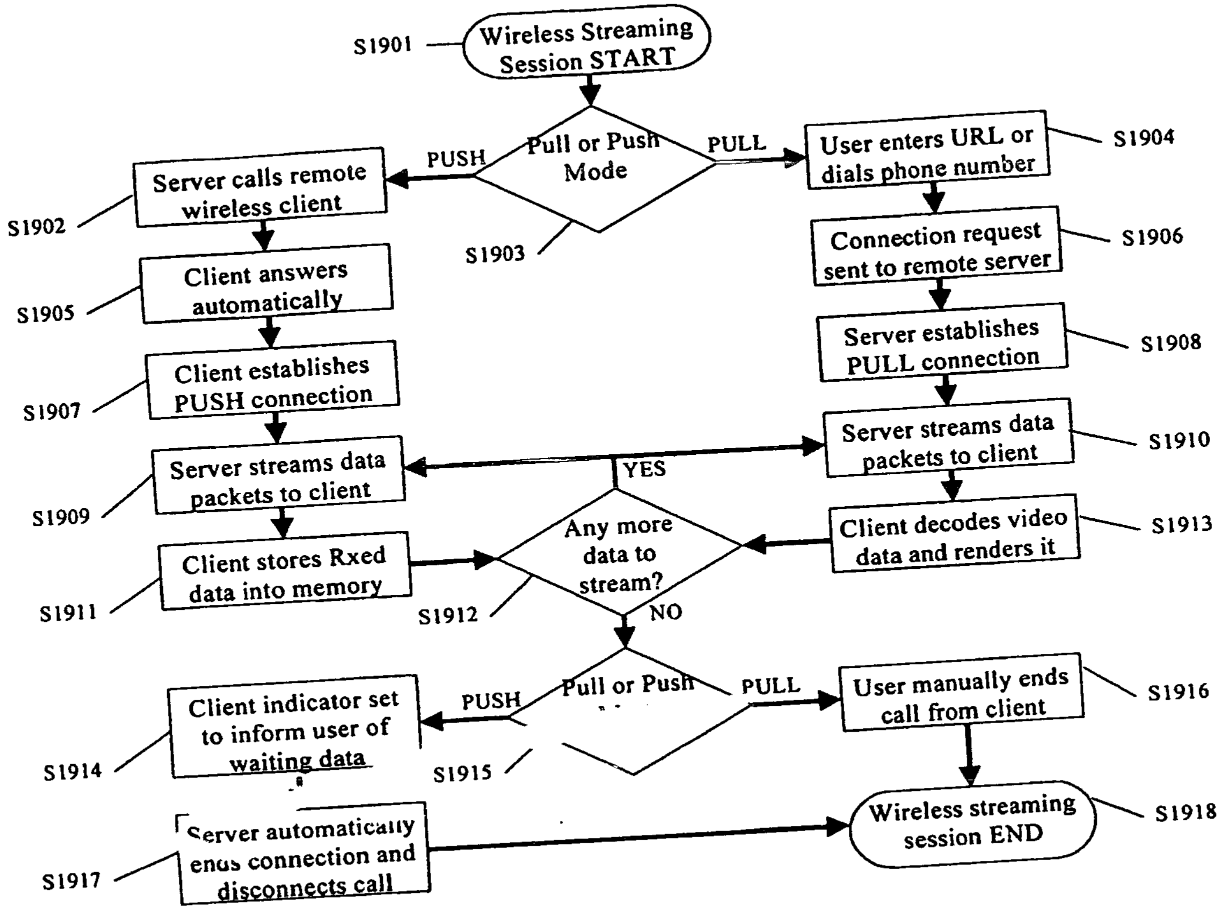

S1901
Wireless Streaming Session START
Pull or Push Mode
S1903
PUSH
PULL
Server calls remote wireless client
S1902
User enters URL or dials phone number
S1904
Client answers automatically
S1905
Connection request sent to remote server
S1906
Client establishes PUSH connection
S1907
Server establishes PULL connection
S1908
Server streams data packets to client
S1909
Server streams data packets to client
S1910
YES
Client stores Rxed data into memory
S1911
Any more data to stream?
S1912
Client decodes video data and renders it
S1913
NO
Pull or Push
PUSH
PULL
Client indicator set to inform user of waiting data
S1914
S1915
User manually ends call from client
S1916
Server automatically ends connection and disconnects call
S1917
Wireless streaming session END
S1918

Figure 40

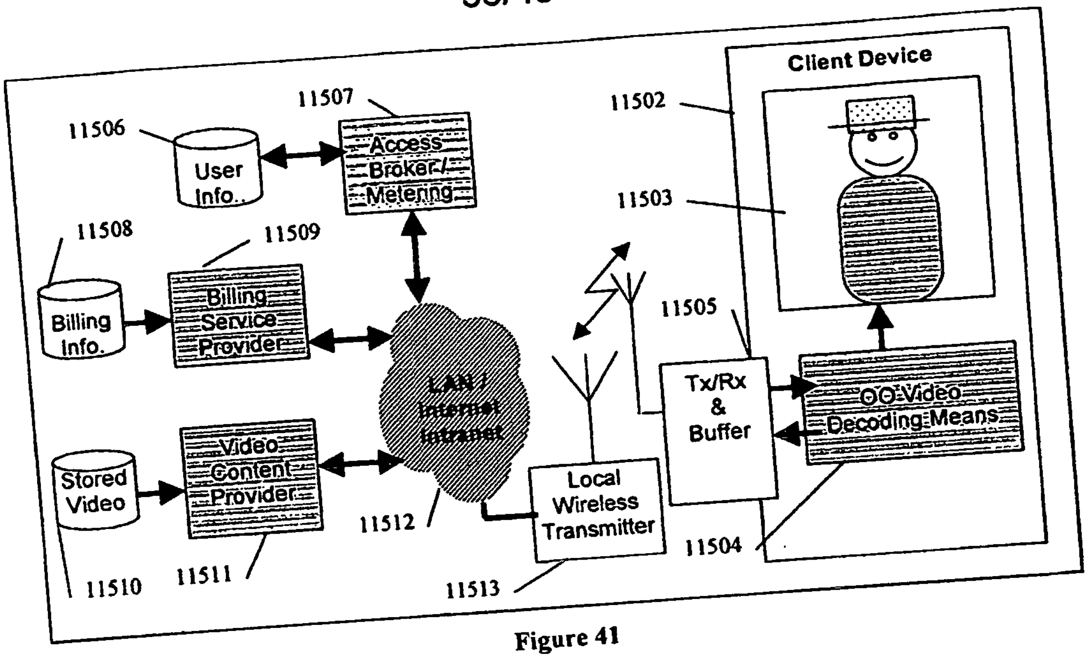

Client Device
11502
11503
11504
11505
11506
11507
11508
11509
11510
11511
11512
11513
User Info.
Access Broker / Metering
Billing Info.
Billing Service Provider
Stored Video
Video Content Provider
LAN / Internet Intranet
Local Wireless Transmitter
Tx/Rx & Buffer
OO Video Decoding Means

Figure 41

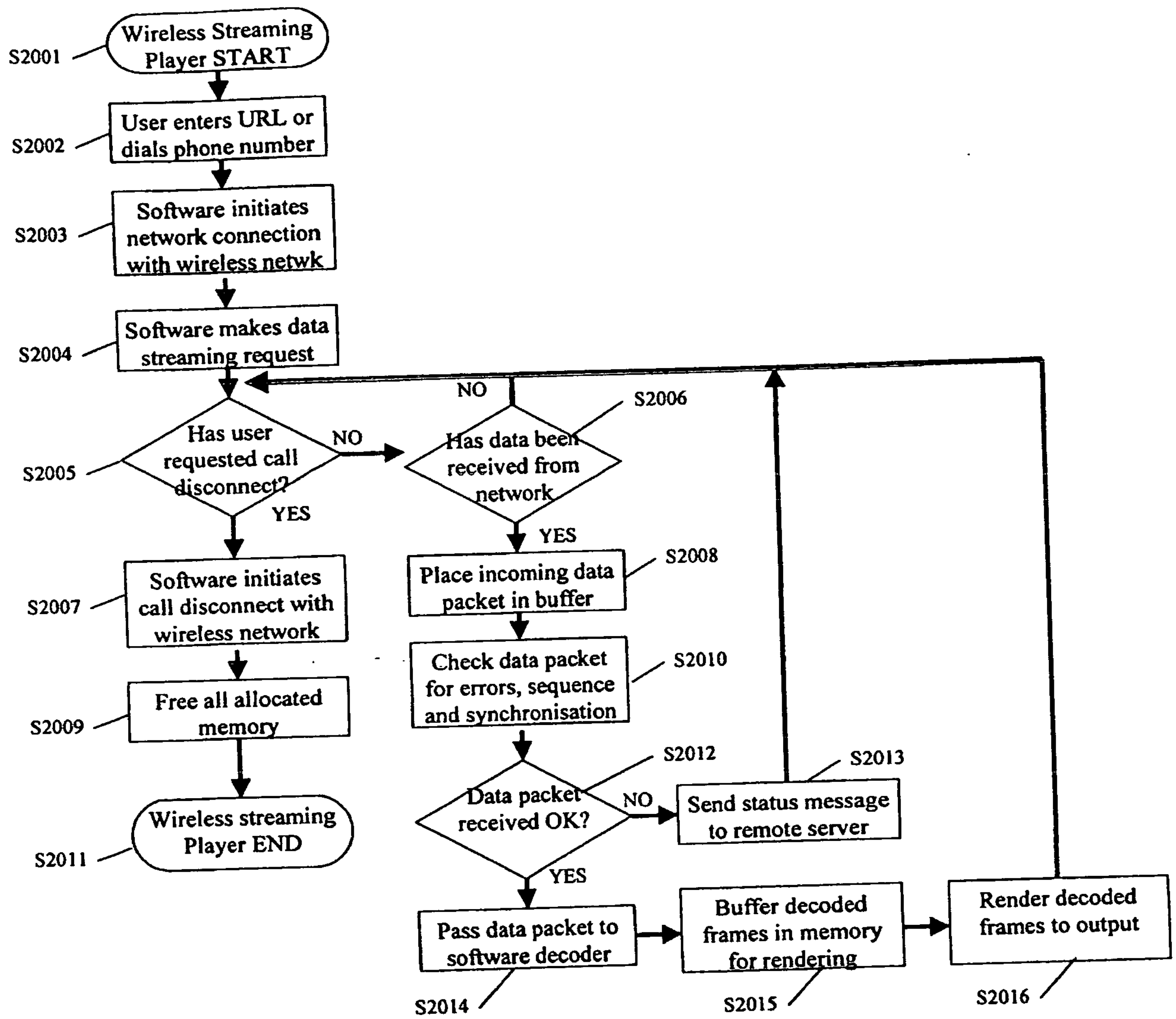

S2001
Wireless Streaming Player START
S2002
User enters URL or dials phone number
S2003
Software initiates network connection with wireless netwk
S2004
Software makes data streaming request
S2005
Has user requested call disconnect?
NO
YES
S2006
Has data been received from network
NO
YES
S2007
Software initiates call disconnect with wireless network
S2008
Place incoming data packet in buffer
S2009
Free all allocated memory
S2010
Check data packet for errors, sequence and synchronisation
S2011
Wireless streaming Player END
S2012
Data packet received OK?
NO
YES
S2013
Send status message to remote server
Pass data packet to software decoder
Buffer decoded frames in memory for rendering
Render decoded frames to output
S2014
S2015
S2016

Figure 42

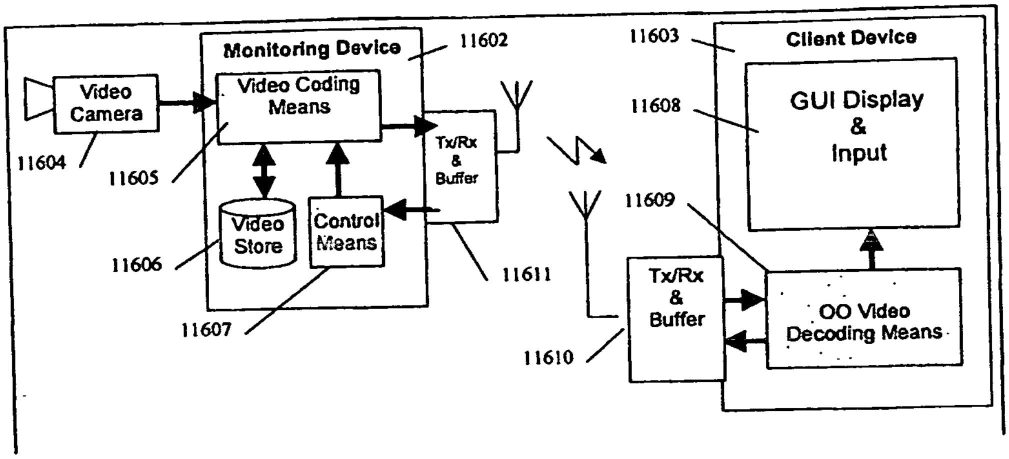

Monitoring Device
11602
11603
Client Device
Video Camera
Video Coding Means
Tx/Rx & Buffer
11608
GUI Display & Input
11604
11605
Video Store
Control Means
11609
11606
11611
Tx/Rx & Buffer
OO Video Decoding Means
11607
11610

Figure 43

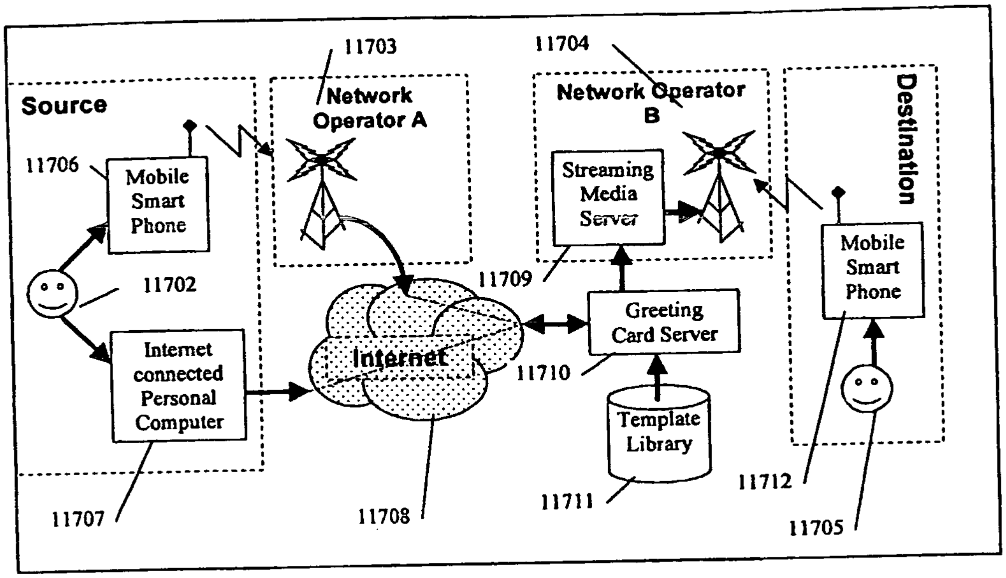

11703
11704
Source
Network Operator A
Network Operator B
Destination
11706
Mobile Smart Phone
Streaming Media Server
Mobile Smart Phone
11702
11709
Greeting Card Server
Internet connected Personal Computer
Internet
11710
Template Library
11712
11707
11708
11711
11705

Figure 44

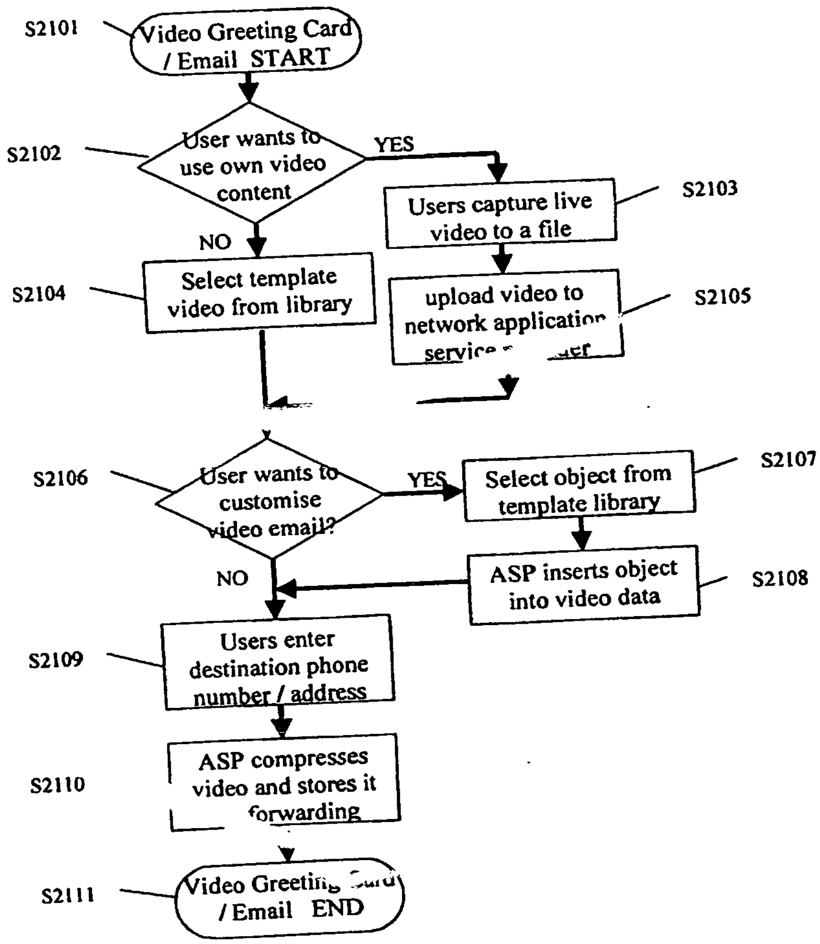

S2101
Video Greeting Card / Email START
S2102
User wants to use own video content
YES
NO
S2103
Users capture live video to a file
S2104
Select template video from library
S2105
upload video to network application
S2106
User wants to customise video email?
YES
S2107
Select object from template library
NO
S2108
ASP inserts object into video data
S2109
Users enter destination phone number / address
S2110
ASP compresses video and stores it
S2111
Video Greeting
/ Email END

Figure 45

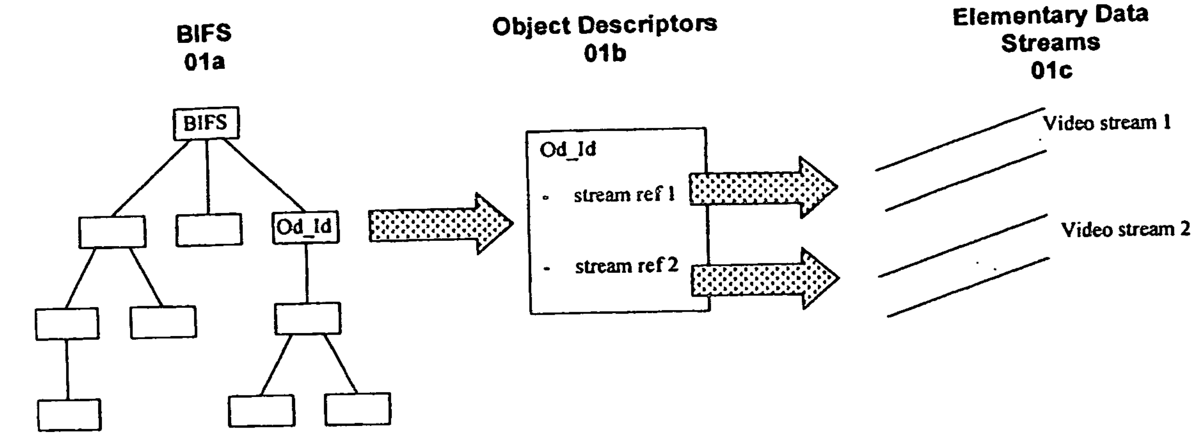

BIFS
01a
Object Descriptors
01b
Elementary Data
Streams
01c
BIFS
Od_Id
Od_Id
- stream ref 1
- stream ref 2
Video stream 1
Video stream 2

Figure 46

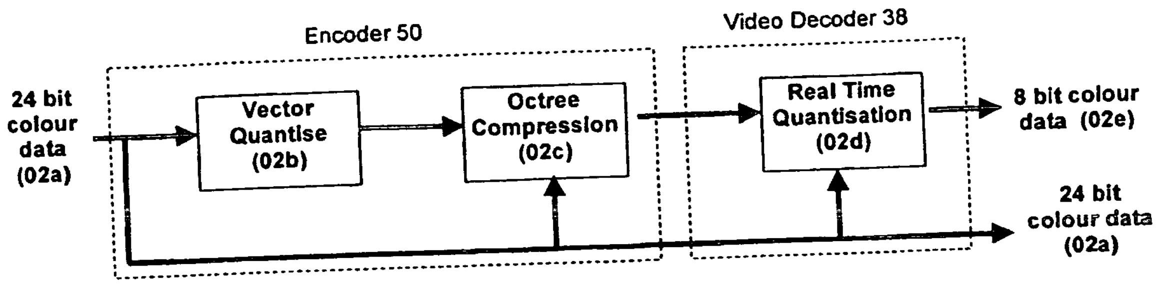

Encoder 50
Video Decoder 38
24 bit colour data (02a)
Vector Quantise (02b)
Octree Compression (02c)
Real Time Quantisation (02d)
8 bit colour data (02e)
24 bit colour data (02a)

Figure 47

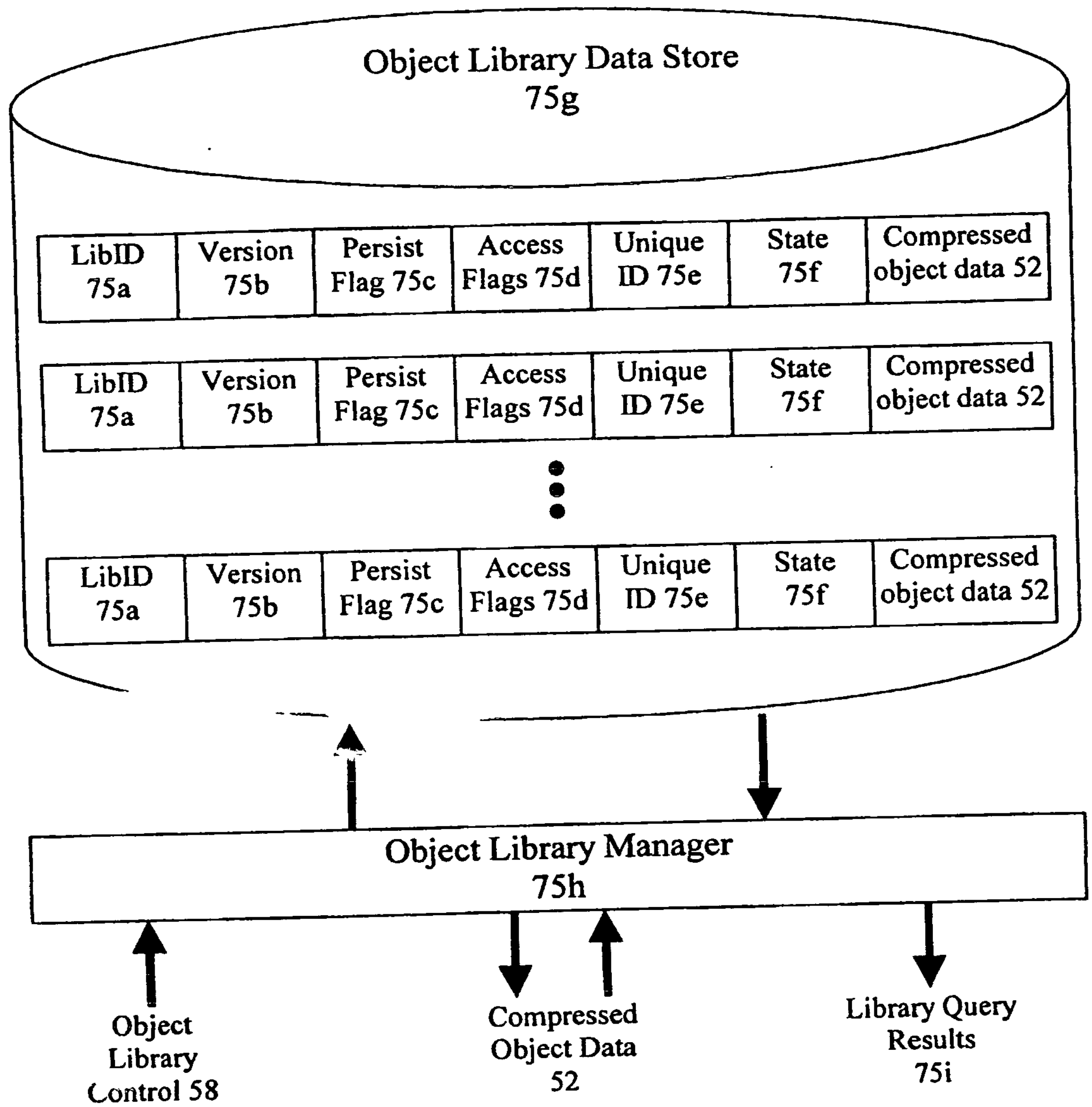

Object Library Data Store
75g
LibID
75a
Version
75b
Persist
Flag 75c
Access
Flags 75d
Unique
ID 75e
State
75f
Compressed
object data 52
LibID
75a
Version
75b
Persist
Flag 75c
Access
Flags 75d
Unique
ID 75e
State
75f
Compressed
object data 52
LibID
75a
Version
75b
Persist
Flag 75c
Access
Flags 75d
Unique
ID 75e
State
75f
Compressed
object data 52
Object Library Manager
75h
Object
Library
Control 58
Compressed
Object Data
52
Library Query
Results
75i

Figure 48

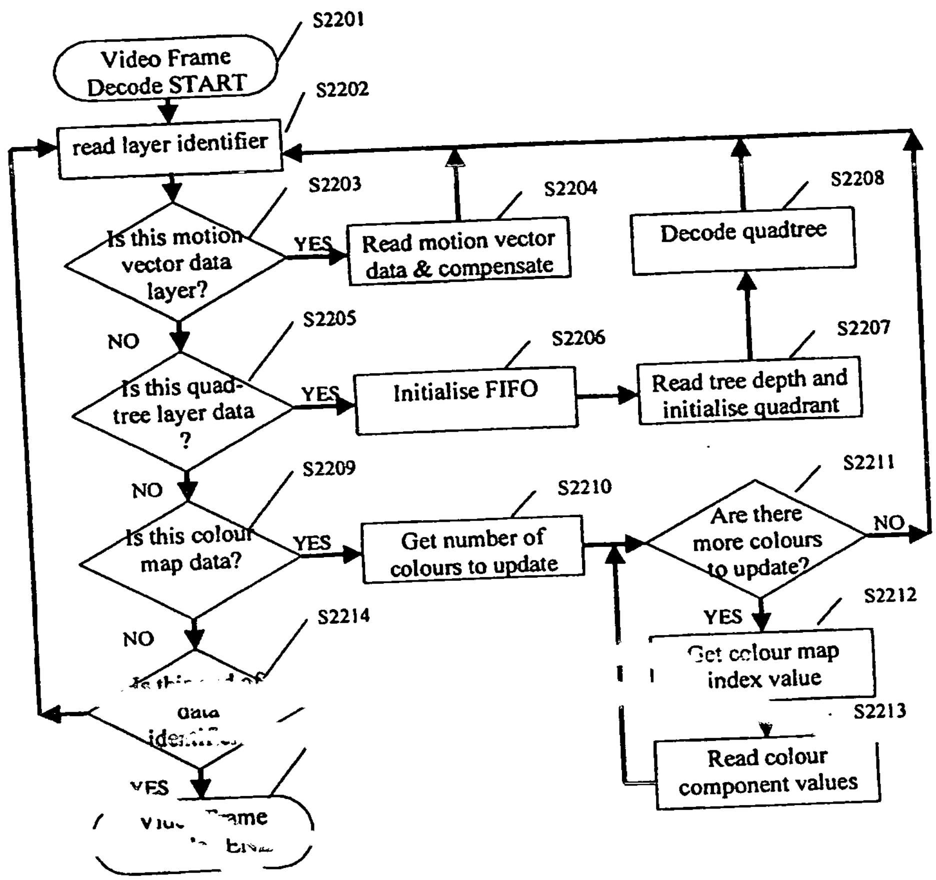

S2201
Video Frame Decode START
S2202
read layer identifier
S2203
Is this motion vector data layer?
YES
S2204
Read motion vector data & compensate
NO
S2205
Is this quad-tree layer data ?
YES
S2206
Initialise FIFO
S2207
Read tree depth and initialise quadrant
S2208
Decode quadtree
NO
S2209
Is this colour map data?
YES
S2210
Get number of colours to update
S2211
Are there more colours to update?
NO
YES
S2212
index value
S2213
Read colour component values
NO
S2214
YES

Figure 49

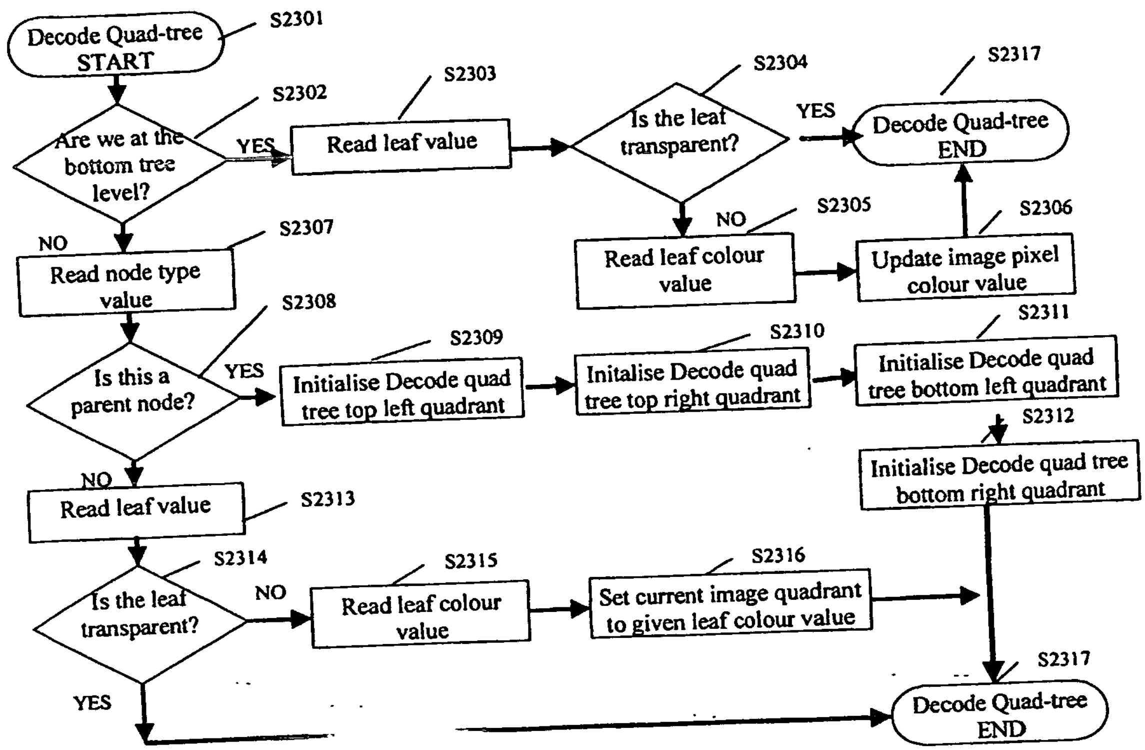

Decode Quad-tree START
S2301
Are we at the bottom tree level?
S2302
YES
NO
Read leaf value
S2303
Is the leaf transparent?
S2304
YES
Decode Quad-tree END
S2317
NO
S2305
Read leaf colour value
Update image pixel colour value
S2306
Read node type value
S2307
S2308
Is this a parent node?
YES
Initialise Decode quad tree top left quadrant
S2309
Initalise Decode quad tree top right quadrant
S2310
Initialise Decode quad tree bottom left quadrant
S2311
S2312
Initialise Decode quad tree bottom right quadrant
NO
Read leaf value
S2313
S2314
Is the leaf transparent?
NO
Read leaf colour value
S2315
Set current image quadrant to given leaf colour value
S2316
YES
S2317
Decode Quad-tree END

Figure 50

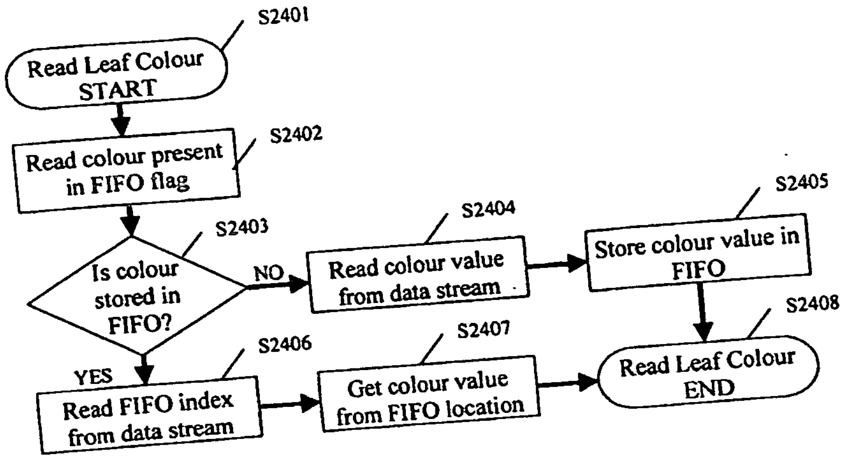

S2401
Read Leaf Colour START
S2402
Read colour present in FIFO flag
S2403
Is colour stored in FIFO?
NO
S2404
Read colour value from data stream
S2405
Store colour value in FIFO
YES
S2406
Read FIFO index from data stream
S2407
Get colour value from FIFO location
S2408
Read Leaf Colour END

Figure 51